REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
05-04-2009		

4. TITLE AND SUBTITLE
Urban Convoy Escort Utilizing a Swarm of UAV's

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Selby, William C. (William Clayton), 1987-

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Naval Academy
Annapolis, MD 21402

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
Trident Scholar Project Report no. 383 (2009)

12. DISTRIBUTION / AVAILABILITY STATEMENT

This document has been approved for public release; its distribution is UNLIMITED

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This project, in conjunction with the naval Academy Research Lab (NRL), evaluated and modified a current UAV control algorithm to perform a security role for military convoys in urban terrain. The desired end state was to provide the simulated military convoy with a constant UAV sensor coverage as the convoy navigated an urban environment.

15. SUBJECT TERMS
urban terrain, Unmanned Aerial Vehicles, (UAV), swarm, control algorithm, military convoy

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**		166	**19b. TELEPHONE NUMBER** *(include area code)*

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

U.S.N.A. --- Trident Scholar project report; no. 383 (2009)

Urban Convoy Escort Utilizing a Swarm of UAVs

by

Midshipman 1/c William C. Selby
United States Naval Academy
Annapolis, Maryland

_____
(signature)

Certification of Advisers Approval

Associate Professor Matthew G. Feemster
Weapons and Systems Engineering Department

_____
(signature)

_____
(date)

Mr. Roger S. Cortesi
Research Engineer, Naval Research Laboratory

_____
(signature)

_____
(date)

Acceptance for the Trident Scholar Committee

Professor Carl E. Wick
Associate Director of Midshipman Research

_____
(signature)

_____
(date)

USNA-1531-2

1. Abstract

How will we keep our troops secure in a hostile and unpredictable environment? Unmanned Aerial Vehicle (UAV) surveillance has wide ranging applications in military convoy operations, as well as other areas of robotics. This project, in conjunction with the Naval Research Lab (NRL), evaluated and modified a current UAV control algorithm to perform a security role for military convoys in urban terrain. The desired end state was to provide the simulated military convoy with constant UAV sensor coverage as the convoy navigated an urban environment.

This research used the NRL multi-vehicle simulator to asses the behavior of the control algorithm under real world conditions. This included using improved vehicle dynamics and real world GPS tracks for convoy routes. The control algorithm was evaluated using performance metrics including the distance between UAVs, distance from each UAV to the convoy, and UAV fuel consumption. The control algorithm was tested in simulation on three scenarios involving a UAV swarm following a military ground convoy. The Basic Navigation scenario simulated a mechanized convoy while the Foot Patrol scenario simulated soldiers on a foot patrol. Lastly, the Obstacles en Route scenario simulated a practical convoy route with constant speed fluctuations.

Based on the data taken from simulations, the control algorithm was modified to provide effective sensor coverage of the convoy in the scenarios. Also, several blending strategies were created to transition between rectilinear and circular control. Specifically, one involving the bearing rate of the convoy relative to the UAVs provides a more secure and low tech form of control than traditional methods. This research identified the limitations of the UMD control algorithm, provided vital data necessary for further development of the controller for field tests, and developed a cumulative design process for future NRL control algorithm investigations.

Keywords: Urban terrain, Unmanned Aerial Vehicles, Swarm, Control algorithm, Military convoy

2. Acknowledgements

I would like to thank primarily my advisors, Prof. Feemster and Mr. Cortesi for their time and efforts devoted to helping me receive this opportunity and their constant guidance. Without their support, I would still be trying to teach myself the fundamentals of coding and controls design. I would also like to thank Dr. Eric Justh for his advice concerning the data analysis and his constant direction and guidance that influenced the path this research followed. I would also like to thank Mr. Dan Robinson for his valuable JAVA knowledge and his continual efforts with the simulator code. Lastly, I would like to thank the USNA faculty, staff and students that have assisted me by answering my questions in their areas of expertise which improved the level of research of this project.

3. Table of Contents

4. List of Figures

5. Introduction

Extensive research is currently being focused on swarm control, making it one of the most exciting and active areas of robotics research. One of the primary reasons for utilizing a swarm for an application is that a single member may not be sufficient to complete an assigned task. A single aerial vehicle may carry all of the sensing equipment necessary for surveillance; however, these vehicles are typically too large and expensive. While a single UAV can cover a specified area using its onboard sensors, a swarm of the same UAVs is able to provide a much larger and overlapping area of sensor coverage. Swarms also have the benefits of increased survivability, reliability, and lower cost due to the ability to decentralize mission specific equipment [1]. Currently, these advantages have led the military to consider introducing aerial vehicle swarms into active service. The Naval Research Laboratory (NRL) in Washington, DC is actively conducting research to determine the effectiveness of unmanned aerial vehicle (UAV) swarms for intelligence, surveillance, and reconnaissance (ISR) missions and particularly in an electronic warfare role.

By adopting a swarm of vehicles for an application, the supervisory control problem becomes more complex. That is, how does one coordinate the efforts of these individualized members in a way to achieve a collaborative objective? To address this problem, formation control algorithms were developed that allow the swarm members to create specified formations for more efficient movement, maneuvering, and target acquisition [1, 2, 3]. Examples of some typical formations are shown in Figure 1.

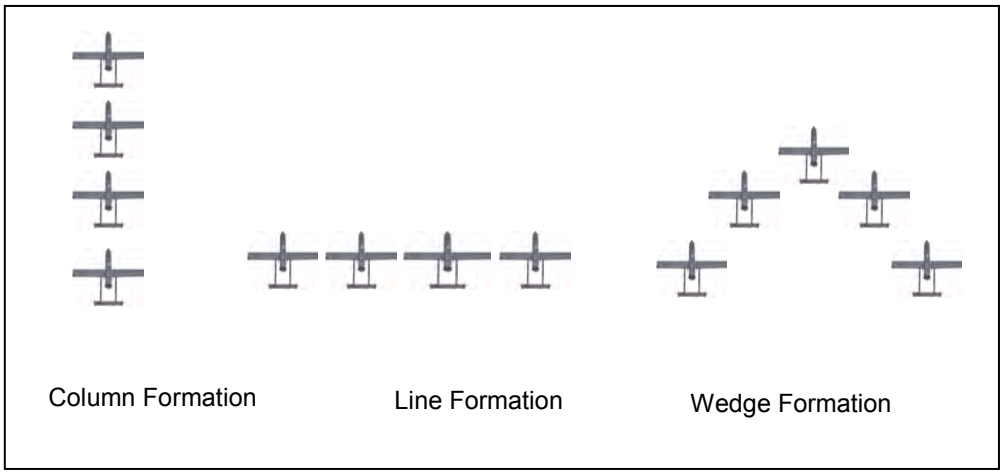


Figure 1: Example Aerial Formations

Select formations also allow individual swarm members to concentrate their sensors on specific areas of responsibility.

Still, many of the existing swarm control algorithms have only been evaluated within either ideal simulation settings (*i.e.*, perfect sensor measurements) or in open experimental environments (*i.e.,* no obstacles). Therefore, it is the primary goal of this research to investigate the ability of an aerial vehicle swarm control algorithm to be utilized within an urban setting subject to real world disturbances. As military operations move into an urban environment, it will be necessary for these aerial swarms to operate reliably in this environment as well. This research looks directly at a convoy escort scenario within an urban terrain (Figure 2), utilizing a swarm of fixed wing aerial vehicles to provide effective sensor coverage.

Figure 2: Ground Convoy in Urban Terrain [5]

An urban environment is particularly hazardous to a military convoy because it offers multiple locations for an enemy to hide as well as the presence of obstacles that can obstruct the path of the convoy. The urban environment drastically limits the maneuvering potential of a military convoy while forcing the convoy into close proximity with the enemy. This close proximity severely limits the US military's technological superiority and weapons capabilities. NRL is especially interested in the ability of a swarm of UAVs to provide electronic jamming for a region around the convoy. It is within this aggressive urban environment that a specific swarm control strategy will be evaluated. Specifically, this research will analyze the ability of an aerial swarm to provide sensor coverage of a military convoy as it moves through a virtual Washington D.C/Annapolis region.

5.1. Current Research

Swarm control algorithms have been motivated to mathematically recreate the behavior of groups of animals such as bees and ants [1, 2, 3, 4] and have been extended to accurately modeling the muscle and behavioral systems of fish [2]. In these biologically inspired examples, the general shape of the swarm is fluid and possesses no predetermined shape (flocking control).

However, other research efforts strive to promote more predetermined configurations where individual vehicles are given specific positions to maintain relative to a leader or neighbor [2, 3] (formation control). Further research has developed control strategies for various objectives including: obstacle avoidance [2, 3], local sensing to achieve group coverage [4], and potential fields for route planning [2].

This proposed research project differs from current research in two key areas. First, this project will investigate a fixed wing aerial swarm control algorithm for suitability within an urban setting, and not focus on individual tasks of the swarm. While having these algorithms for optimal task assignment and route planning are important, they are of little use if the swarm is unable to operate effectively in urban terrain. Rather than focusing on behaviors of specific vehicles in the swarm, this research will critically assess the behavior of the entire UAV swarm using a specific control algorithm to determine if UAVs are even capable of performing urban centric missions. Secondly, many of the algorithms proposed in the existing literature are verified using ideal simulations. Factors such as weather, communications or sensor loss, and the presence of obstacles are omitted. With the advanced capabilities of the NRL multi-vehicle simulator, this research will introduce these realistic disturbances in the simulated environment and observe how the control algorithm is able to handle the more realistic variables.

5.2. Control Algorithm for Analysis

Though many swarm control algorithms exist, the primary controller under consideration for this research project was designed by Dr. Eric Justh and Dr. P.S. Krishnaprasad from the University of Maryland [12] for control of a swarm of fixed wing UAVs. Specifically, this control law was selected over others for two primary reasons: 1) the algorithms include such aerial vehicle constraints as constant speed and 2) the ability of the control strategy to produce

both transiting (rectilinear) and loitering swarm formations. However, the control algorithm of [12] has only been simulated under ideal conditions. Currently, NRL is considering expending much effort in the development of an experimental aerial swarm based around this control strategy; thus, the algorithm must initially be evaluated in detail in order to be determined effective (this evaluation represents one of the focus points of this project). Drs. Justh and Krishnaprasad designed their algorithm to perform a few basic functions which make it applicable to multiple scenarios. Specifically, their algorithm was biologically inspired. They wished to model three basic characteristics of biological swarms: common orientation, cohesion, and non-collision [12]. Their algorithm achieves these behaviors through the utilization of a geometric framework which creates a simple and straightforward representation of the constraints on the system. A weighted average of the three gains, η, α, μ, is utilized to control the overall swarm formation depicted in Figure 3.

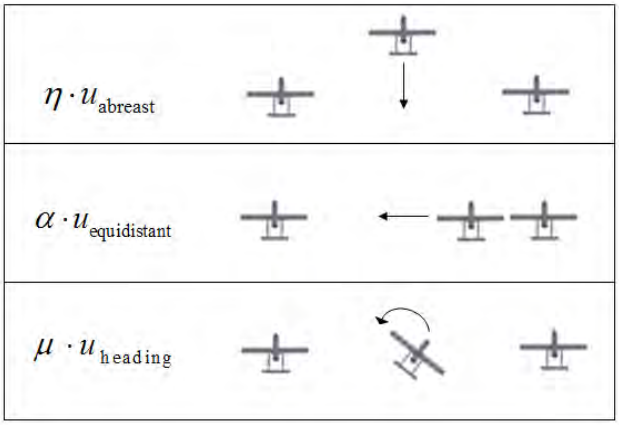


Figure 3: Potential Field Behavior Illustration

In addition to evaluating the control strategy of [12] within realistic settings, an investigation to determine how to transition between rectilinear (rapid convoy movement) and loitering (slow or stationary convoy) swarm movement will be performed. Currently, the controller of [12] does

not state an explicit strategy for this transition. Therefore, a blending algorithm must be developed that will allow the swarm formation to effectively transition between these linear and circular formations.

5.3. Problem Statement

The primary objective of this research is to provide a comprehensive analysis and modification of the UMD control algorithm [12] when applied to a swarm of unmanned aerial vehicles for automated urban convoy escort. The performance of the UMD controller will be evaluated against the following primary performance metrics (Figure 4):

Metric	Tactical Significance
Distance between UAVs	Too close => risk of collision Too far => communication loss
Distance from Convoy	Too far => jamming and sensors become ineffective
Fuel Consumption	More maneuvering = more fuel used = shorter mission duration

Figure 4: Performance Metrics and their Tactical Significance

In addition, a blending algorithm to augment the control design of [12] that will allow the swarm to alter shape depending on such factors as convoy speed and organization will be investigated. Also, modification of the algorithm to support a convoy if it splits into two distinct groups will be investigated. These scenarios will be discussed with more detail in the following sections.

5.4. Task Overview

This section provides a brief overview of the primary tasks that are necessary to successfully research the convoy escort problem presented in the problem statement utilizing the UMD control law.

- Investigation of UMD Controller [12]

Prior to the implementation of this controller into the NRL multi-vehicle simulator, it is necessary to acquire a solid understanding of the control strategy the authors used in designing this controller. Specifically, I analyzed the controller's design process and then implemented the algorithm in MATLAB. This task familiarized me with all the required measurements, calculations, and tunable control parameters involved.

- Generation of Convoy Trajectory

In an effort to make the simulation of a foot or mechanized convoy as realistic as possible, I generated sample convoy trajectories throughout the Washington D.C. metropolitan area. GPS receivers logged the trajectories of the convoy elements through the city. These trajectories involved both walking (foot patrol) as well as street navigation in an automobile (mechanized convoy) to simulate military convoy scenarios. In order to do this, I co-created a GPS route module that was inserted into the NRL multi-vehicle simulator. This allowed me to inject the routes I created by moving through the city.

- Implementation into the NRL Multi-Vehicle Simulator

This task involved modifying the existing coding of the UMD control law in the NRL multi-vehicle simulator in order to support the parameter modification necessary for this project. Parameter modification is the process of changing the values of η, α, μ, r_0 and analyzing the resulting swarm behavior. This process allows insights to be drawn about the relative effects of each of the parameters. It also required the creating of code in the simulator to export specific vehicle data as well as the creation of a MATLAB script to post-process this data.

- Evaluation of the UMD controller in the Urban Environment

The evaluation consisted of running the swarm through ever more complex scenarios and observing the results based on the performance metrics. During these scenario simulations, specific data was taken and the swarm's performance was measured based on multiple initial conditions. The objective was to determine in which scenarios, if any, the UMD controller was effective and to identify the areas of the controller that need to be re-evaluated in order to be successful in the future. The information collected allowed me to make specific recommendations supported by realistic data.

- Development of Blending Algorithm:

This task consisted of formulating a blending strategy that allowed the swarm to smoothly transition between two forms of the control law depending on such factors as convoy speed and distance. While the ability for the swarm to create rectilinear and circular formations is already proven, there was no current focus on how to transition between the formations and under what circumstances. The rectilinear and circular control laws work well under ideal situations, but adding the complexities of a real world mission with varying convoy speeds requires a method to use varying amounts of the rectilinear and circular control outputs in order to improve swarm performance.

5.5. Controller Investigation

This swarm controller was designed to replicate a formation of animals. The author's primary design objectives were to use automatic control to avoid collisions between vehicles, maintain formation cohesiveness, be robust to loss of individuals, and scale favorably to large swarms

[12]. Their equations are suitable for control of small UAVs for *n* number of vehicles. The control equation is reproduced as follows:

$$\dot{\mathbf{r}}_j = \mathbf{x}_j, \quad \dot{\mathbf{x}}_j = \mathbf{y}_j u_j, \quad \dot{\mathbf{y}}_j = -\mathbf{x}_j u_j$$

$$u_j = \frac{1}{n}\sum_{k\neq j}\left[-\eta\left(\frac{\mathbf{r}_{jk}}{|\mathbf{r}_{jk}|}\cdot\mathbf{x}_j\right)\left(\frac{\mathbf{r}_{jk}}{|\mathbf{r}_{jk}|}\cdot\mathbf{y}_j\right)-\alpha\left[1-\left(\frac{r_0}{|\mathbf{r}|}\right)^2\right]\left(\frac{\mathbf{r}_{jk}}{|\mathbf{r}_{jk}|}\bullet\mathbf{y}_j\right)+\mu\mathbf{x}_k\cdot\mathbf{y}_j\right] \qquad \textbf{[1]}$$

where the following variables are defined as:

- $\mathbf{r}_{jk} = \mathbf{r}_j - \mathbf{r}_k$ which is the distance between any two UAVs
- $\mathbf{x}$ is the unit tangent vector to the UAV trajectory
- $\mathbf{y}$ is the unit normal vector to the UAV trajectory
- u is the curvature or steering control

These equations above control the position and approximate orientation of the UAV. In the steering control equation, the term involving μ (mu) aligns the heading directions of the vehicles, the α (alpha) gain is used to control the distances between the UAVs as regulated by the r_0 distance, and the η (eta) term is used to keep the vehicles abreast of each other. Each of these terms emulates a biologically plausible behavior. These specific gains are the primary focus of this research. The gains effectively create a weighted average of the three behaviors to produce a steering command for the vehicle based on its position and heading relative to all other vehicles in the swarm. Based on the scenarios and performance metrics, these gains will be modified extensively to create an algorithm tuned for specific objectives in an urban environment.

6. Investigation of UMD Control Law

6.1. Algorithm Derivation

The control algorithm can be given a mechanical interpretation by considering the motion of a charged particle in a magnetic field. The equations of motion are derived by formulating a Lagrangian using the kinetic and potential energies, from which Euler-Lagrange equations are derived. The Lorentz force law for a charged particle in a magnetic field is given by $K = \frac{q(\dot{\mathbf{r}} \times \mathbf{B})}{c}$ where q is the particle's magnetic charge, $\mathbf{B}$ is the magnetic field and c is the speed of light. Refer to Figure 6 for the coordinate system used.

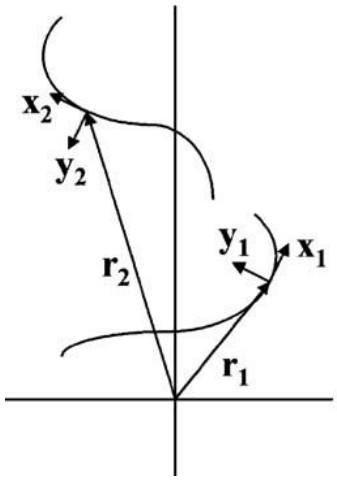


Figure 5: Algorithm Derivation Coordinate Frame [12]

If $F = m \cdot \ddot{\mathbf{r}} = \frac{q \cdot (\dot{\mathbf{r}} \times \mathbf{B})}{c}$ and the magnetic field $\mathbf{B}$ is perpendicular to the plane of motion $\mathbf{B} = (0,0,B_z)^T$ then $\ddot{\mathbf{r}} = \frac{q \cdot (\dot{\mathbf{r}} \times \mathbf{B})}{mc}$. If $u = -\frac{qB_z}{mc}$ and we define $\mathbf{x} = \dot{\mathbf{r}}$ and $\mathbf{y} = \mathbf{x}^{\perp}$, then we obtain $\dot{\mathbf{x}} = u\mathbf{y}$ and $\dot{\mathbf{y}} = -u\mathbf{x}$. This simplifies into $\begin{bmatrix}\dot{\mathbf{x}} & \dot{\mathbf{y}}\end{bmatrix} = \begin{bmatrix}0 & -u \\ u & 0\end{bmatrix}\begin{bmatrix}\mathbf{x} & \mathbf{y}\end{bmatrix}$ and the Planar Frenet-Serret equations of motion are derived: $\dot{\mathbf{r}} = \mathbf{x}$, $\dot{\mathbf{x}} = \mathbf{y} \cdot u$ and $\dot{\mathbf{y}} = -\mathbf{x} \cdot u$. As seen in Figure

5, **x** is the unit tangent velocity vector, **y** is the unit normal velocity vector, and *u* is the curvature of steering control which is calculated by the control algorithm. Equation 1, the control algorithm is then used to determine *u*, the steering control input, based on the interaction between the vehicles.

6.2. MATLAB Simulations

6.2.1. Rectilinear and Circular Control

From this basic understanding of the equations of motion and the control algorithm, I was able to code a simple simulation in MATLAB. The basic flow of information is shown in Figure 6 where x and y are the vehicle's position, θ is the heading of the vehicle, and *u* is the control law output. This code can be seen in Appendix 1 where it is explained in further detail.

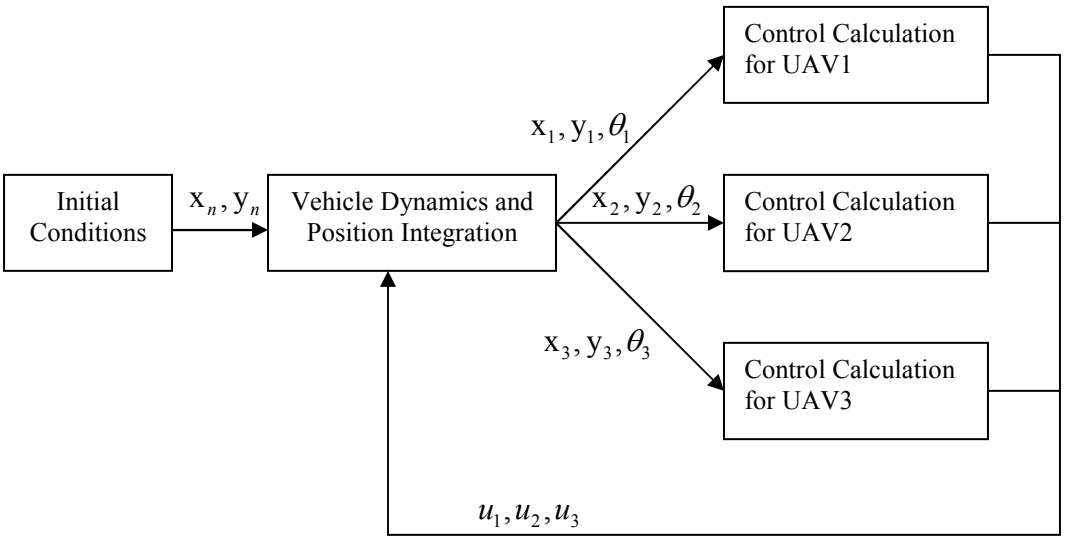


Figure 6: MATLAB Code Flow Chart

The simulation loop in the code calculates the total *u* for each vehicle and then exits the loop. The control calculation is performed once per time step *dt* and the process continues until the simulation time expires. This produced plots similar to Figures 7-9, based on initial

conditions. Figure 7 shows the trajectories of the vehicles from random initial positions and headings and finish in close proximity travelling in a similar direction.

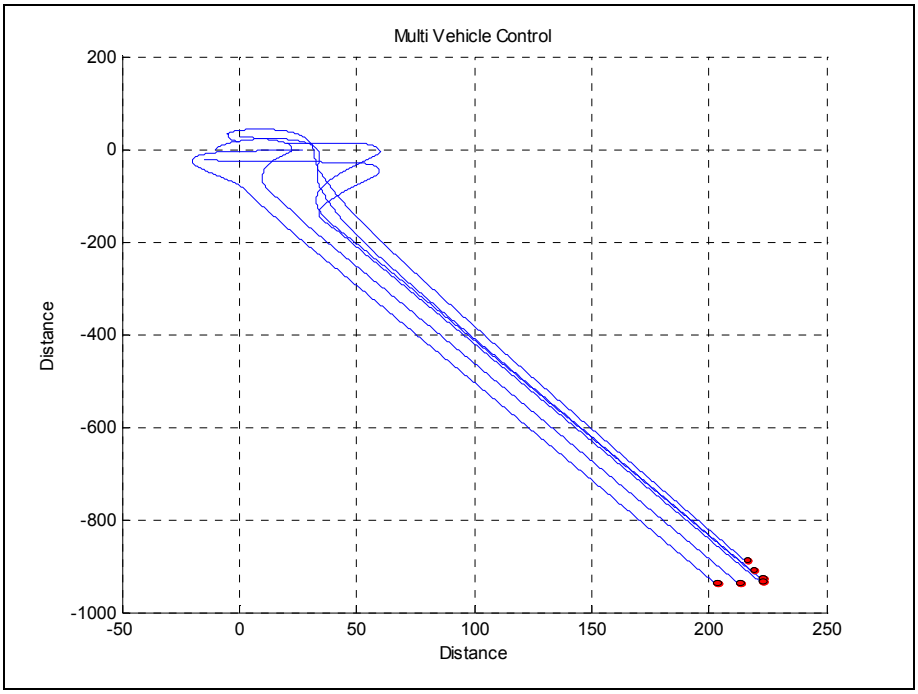


Figure 7: Vehicle Positions

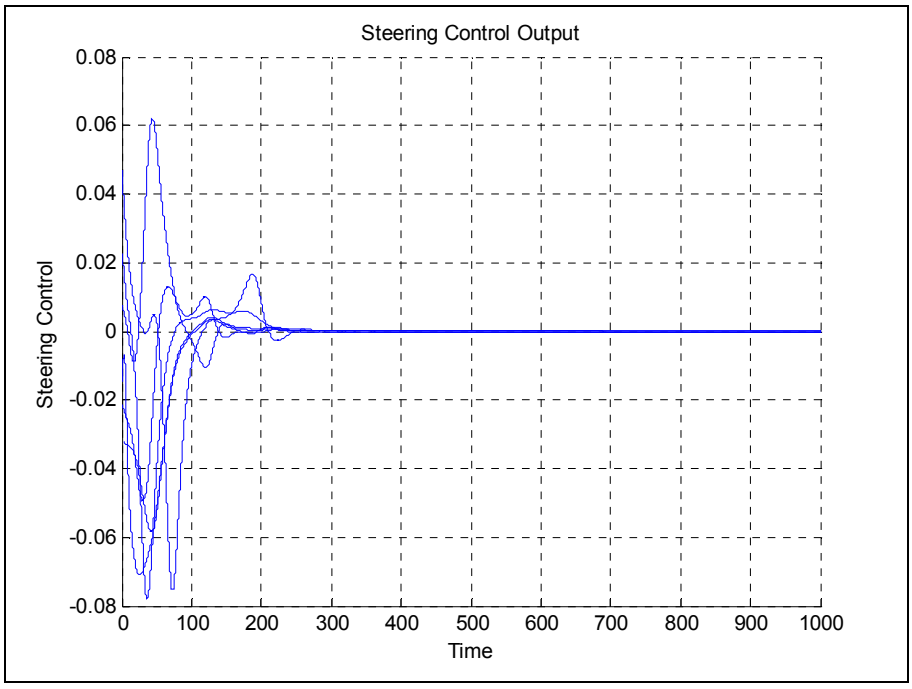


Figure 8: Vehicle Steering Control Output

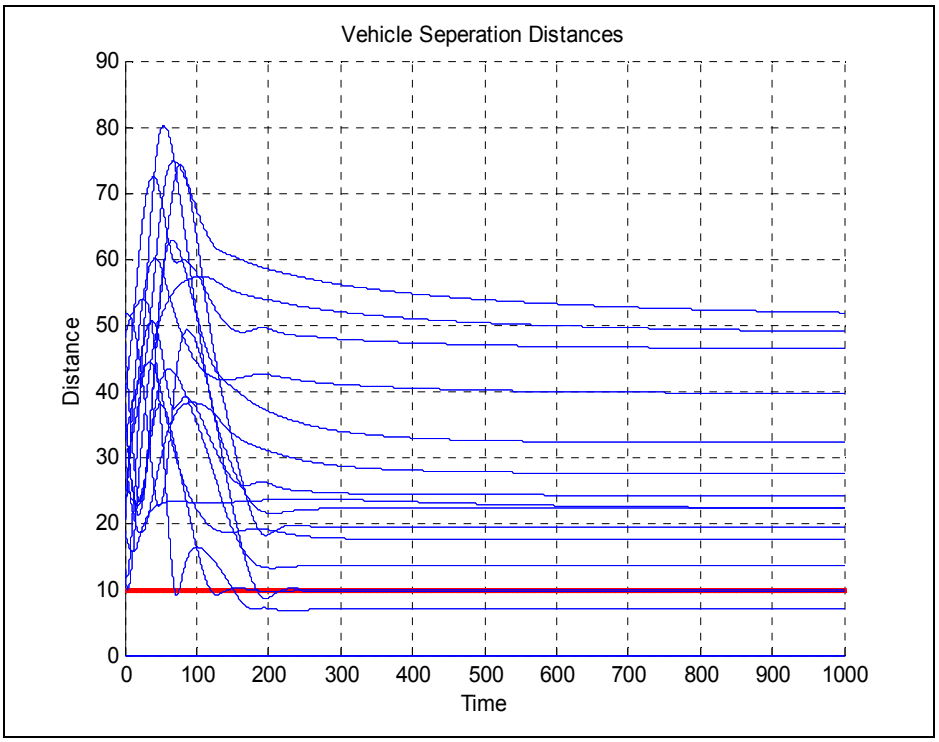


Figure 9: Intra-Vehicle Separation Distances

Looking at the plot of the steering control u in Figure 8, it is clear that the vehicles had to do a lot of maneuvering in the beginning to get aligned and headed in the right direction, but once this was done, steering command values dropped to about 0 as no further adjustments were needed. This is also seen on the vehicle separation plot, Figure 9. As time continues, the vehicles find their steady state positions and the intra-vehicle distances remain generally constant and approach r_0 as indicated by the red line.

Next, the control law was modified in the code to support the circular control law and similar plots were created. This was done simply by removing the μ term from the calculation of the control. This creates the circling motion by forcing the vehicles to be equidistant as well as perpendicular to the baseline between them and removing the behavior which controls heading alignment. Implementing this form of the control law created the plot in Figure 10.

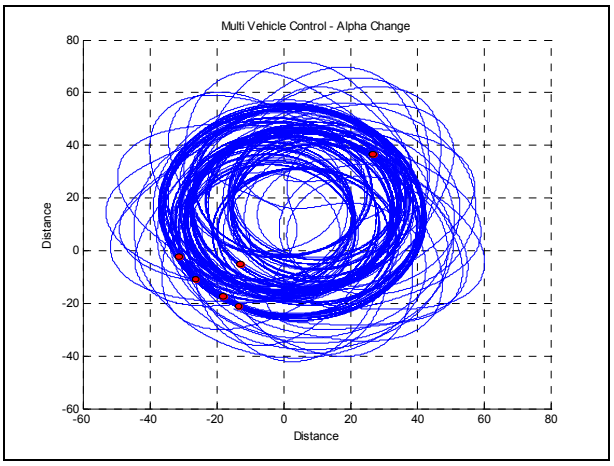


Figure 10: Vehicle Trajectory Using Circular Control Law

6.2.2. Parameter Modification

Next, the code was modified to allow for mid-simulation parameter modification. This was done by implementing an "if" statement inside the simulation loop that switched between two different parameter values halfway through the simulation. This way, it was possible to run the simulation and change individual gain values during the simulation to see the vehicles' responses. Plots of these results can be seen in Appendix 2. The first plot is the simulation run with base gain parameters and no mid-simulation modification (Appendix 2.1). At first, η was changed as shown in Appendix 2.2. When η was made smaller by a factor of 10, its impact on the behavior of the swarm was reduced and therefore the vehicles begin to converge. This is because the heading alignment behavior and the separation distance behavior are primarily controlling the vehicles. However, when η is made larger by a factor of 10, its behavior dominates as seen in Appendix 2.2. Of note is the corkscrew that two vehicles perform right after the switch. This may be due to the fact that these two vehicles were ahead of a vehicle in close proximity. Thus, when the parameter was changed, they were forced to circle the vehicle behind them, keeping all three vehicles abreast, and allowing the vehicle to catch up. Once this

maneuver was done, these two vehicles continued in the general forward direction the same as the rest of the vehicles.

Plots in Appendix 2.3 depict the behavior of μ, the heading alignment gain. In the first plot, η is reduced by a factor of ten and it is clear that heading alignment is not being maintained, visible by the oscillatory motion of a few of the central vehicles. However, when the emphasis is placed on μ, the vehicles immediately find a common direction and their forward paths appear to be perfectly parallel to each other. These plots clearly demonstrate the importance of μ. This information may be particularly useful when analyzing fuel consumption. As the vehicles oscillate, they cover more distance and maneuver more often, both of which consume more fuel. However, if μ is ramped up, it appears that all vehicles will take a straight path, and thus conserve energy.

Third, the α parameter was adjusted as shown in Appendix 2.4. This parameter controls the ability of the vehicles to maintain a constant spacing, as defined by the fourth parameter, r_0. As α is decreased, some vehicles move away from each other while some remain extremely close. This is contrasted with the following plot, when α is increased. Here it is clear that all of the vehicles begin to converge immediately and as the simulation continues, they approach a very tight formation. Looking closer, r_0 is set at 20 units and it appears that several vehicles are closer than this minimum distance which could create collisions in the real world. Since the control law is calculated based on each vehicle's position relative to every other vehicle's position and not just the vehicle in closest proximity, it is possible that by increasing α drastically, one can nearly eliminate any factor of safety designed by the r_0 distance which could lead to mid-air collisions.

Finally, r_0, the separation distance, was adjusted and the predicted behavioral response was observed as shown in Appendix 2.5. As r_0 is drastically decreased, the majority of vehicles violently converge upon one another. As r_0 is increased, the vehicles immediately begin to spread out. This process of parameter modification was repeated on the circular law as well. Similar results were seen as the parameters were modified but in a less clear manner due to the fact that the effects of the parameters on the trajectories are more coupled. However, from combining these results and analyzing the output, the influence of each parameter becomes visually evident and serves as a strong foundation for further comprehension of the control algorithm parameters.

6.2.3. Following Waypoints

In the scenarios, a vehicle representing the military convoy sends out its position via a GPS unit that the vehicle swarm is able to receive. Therefore, the next step in the code progression was to implement a "beacon" or waypoint for the vehicles to respond to. The waypoint was initialized with an x and y position as vehicle number one. To prevent the waypoint from moving, it was skipped over during the integration and position update loop in the code as is done in NRL's multi-vehicle simulator. This proved acceptable when the swarm was kept to smaller numbers as seen in Figure 11.

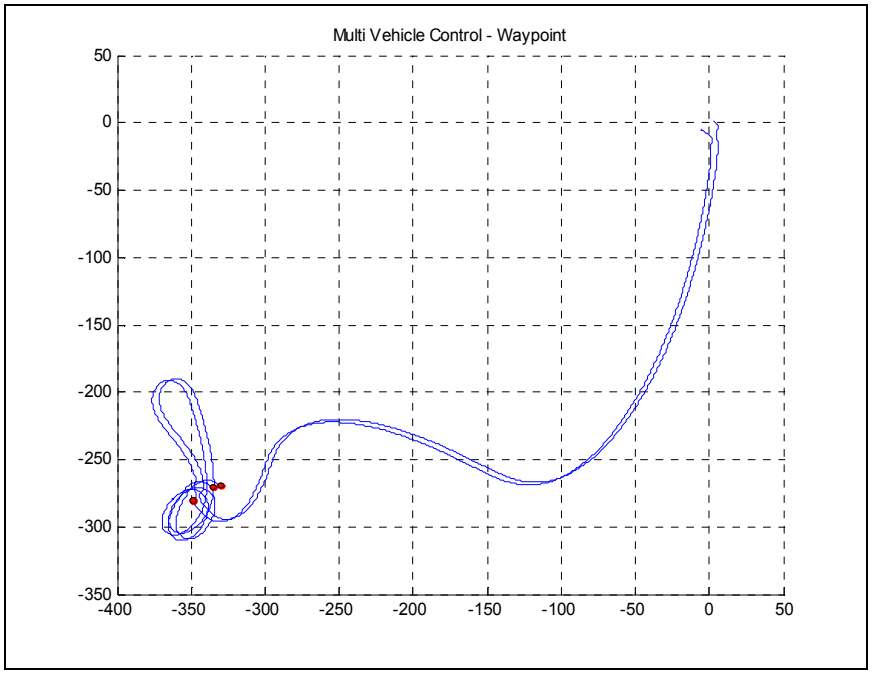


Figure 11: Two Vehicles Responding to One Waypoint

However, when the swarm was increased to three vehicles, the vehicles did not respond to the waypoint and instead moved forward in a different direction. After analyzing this data, it was evident that the there was an issue with weighting the control calculation from the vehicles to the waypoint. Initially, this was solved by weighting the steering command between the vehicles and the waypoint in order to give it a greater value over the intra-vehicle control calculations. As the swarm size grew from three to six, the weighting amount grew exponentially. Instead, the code was modified to include the ability for each vehicle to compute its individual heading direction to the waypoint. This code is explained in further detail in Appendix 1

Also in Appendix 1 is the ability of the swarm to follow a set of waypoints. If one vehicle gets within a certain distance of the waypoint, the current waypoint is removed and the next waypoint is used for all future control calculations. The code was also modified so that the

waypoint locations would update on a per vehicle basis, rather than for every vehicle at one time. These modifications produced plots such as Figure 12.

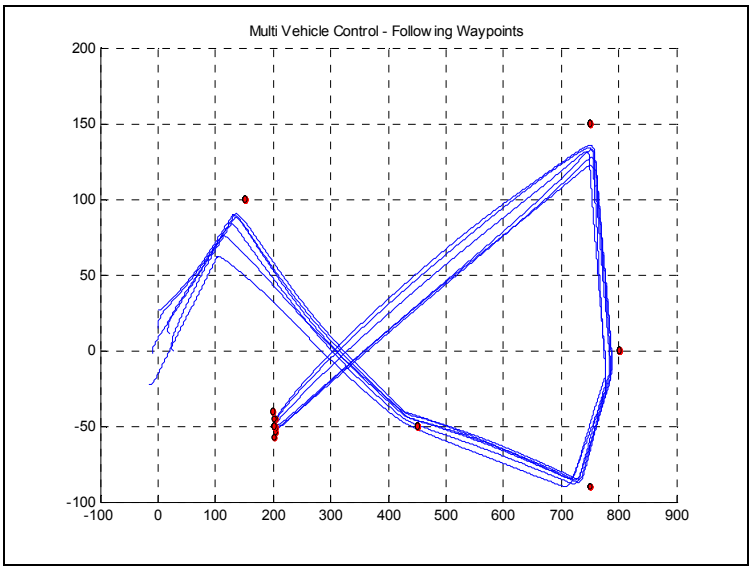


Figure 12: Swarm Moving Between Waypoints Using Rectilinear Control

6.2.4. Blended Control

To conclude the MATLAB initial analysis, a basic linear blending method was established as well as the ability for the vehicles to loiter at specified waypoints. The final code incorporating all of these changes can be seen in Appendix 1 along with a detailed explanation of the code. The effects of this blending process can be seen when comparing the non-blended control in Figure 13 with the blended control in Figure 14. For the non-blended control plot, the vehicles initially start using pure rectilinear control, switch to pure circular after the first waypoint, and then switch back to pure rectilinear after loitering at the second waypoint. This is a hard switch and there was no combination of the rectilinear and circular control methods.

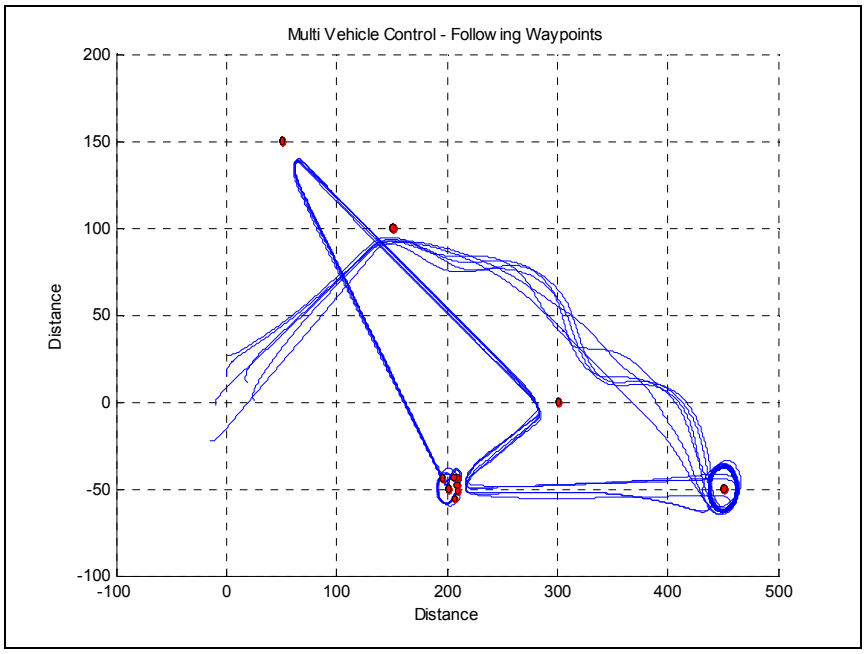


Figure 13: Swarm Response without Blending

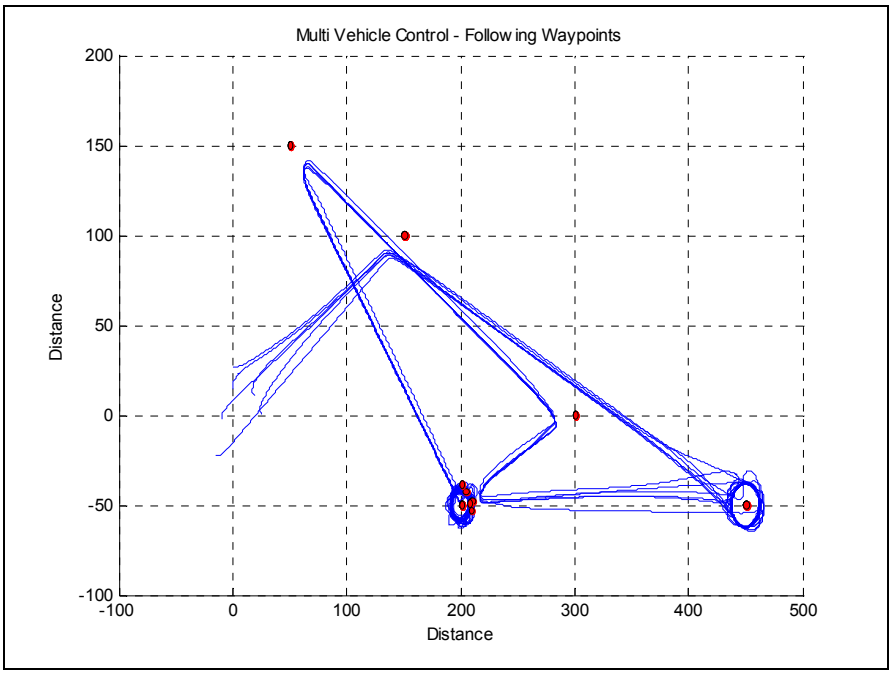


Figure 14: Swarm Response with Blending

It is clear that the blending algorithm is a more efficient means of behavior control for the swarm and results in much smoother movement between waypoints. Further research in the multi-vehicle simulator was devoted to analyzing the effects of a non-linear progression as well as what the cut off distances for the control transition.

6.2.5. Extension into JAVA

The NRL multi-vehicle simulator is a JAVA based program, so the next step of the project involved converting the MATLAB code into working JAVA code in order to have a better understanding of the complex code in the NRL simulator. Several JAVA classes were created and can be seen in detail in Appendix 3. To analyze the data, a JAVA class was created to write the position values of the vehicles into a text file and MATLAB was used to analyze the data and produce plots. With the same initial conditions, both the MATLAB (top) and JAVA (bottom) simulations produce the same results as seen in Figure 15.

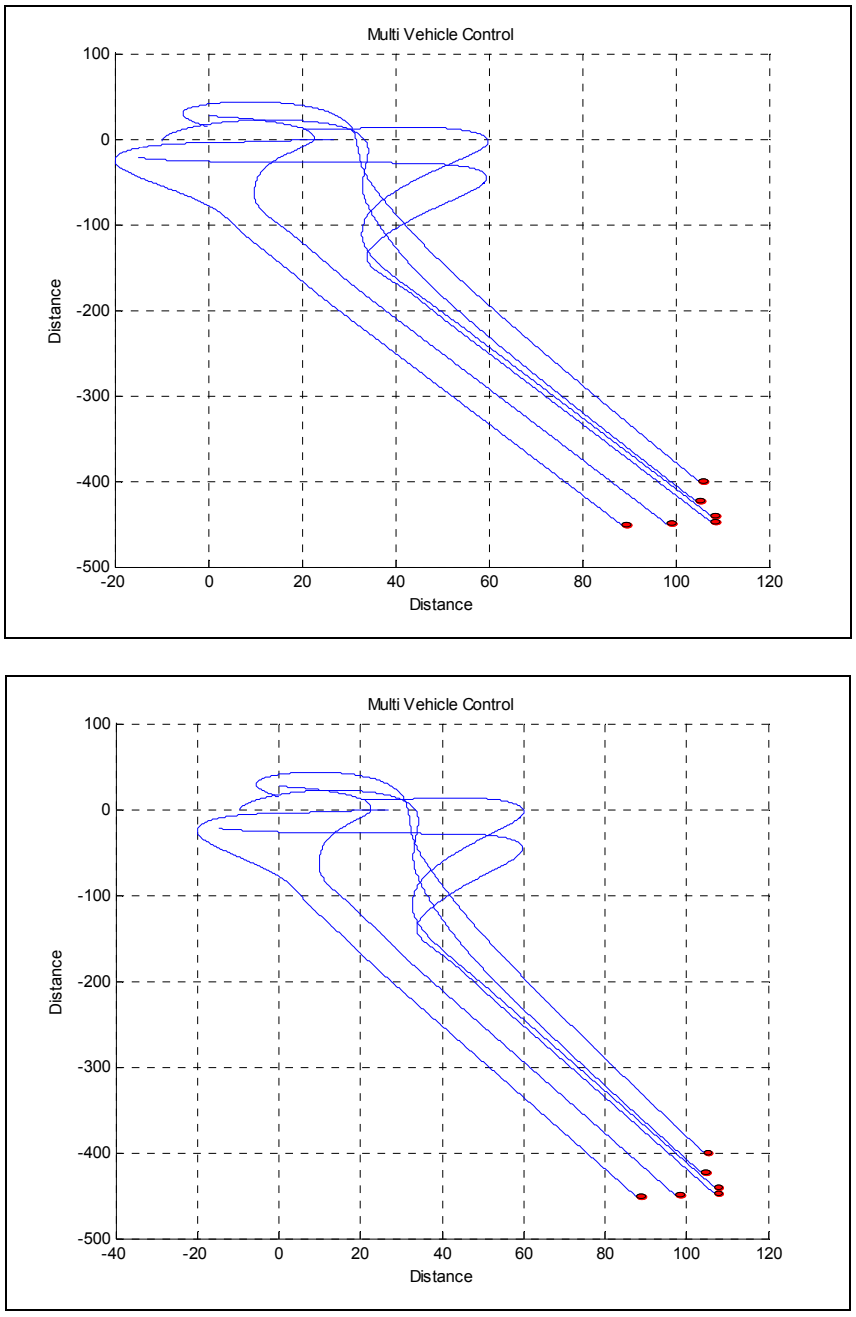


Figure 15: Comparison of MATLAB and JAVA Simulation Outputs

Using the unique style of object oriented coding that is characteristic of JAVA, the more complex MATLAB functions were incorporated into the JAVA simulation package. Since the control calculations were done in a separate class, modification was straightforward and the code enabled the user to easily switch between circular, rectilinear, or blended control. The ability to

calculate the heading from each vehicle to a specified waypoint was added as evident in the calcWptControl method in Appendix 3.5. Since the rectilinear and circular control calculations were done in separate methods (calcControlRect and calcControlCirc in Appendix 3.5), the implementation of a blending control strategy was easily done. This can be seen in the blendControl method in Appendix 3.5. Lastly, the waypoint class was modified to support waypoint loitering on a Boolean logic basis or on a distinct time amount basis. This can be seen in the checkAndSwitch method in Appendix 3.4. With this addition, simulations where the vehicles loiter around certain waypoints for varying amounts of time were run Figure 16.

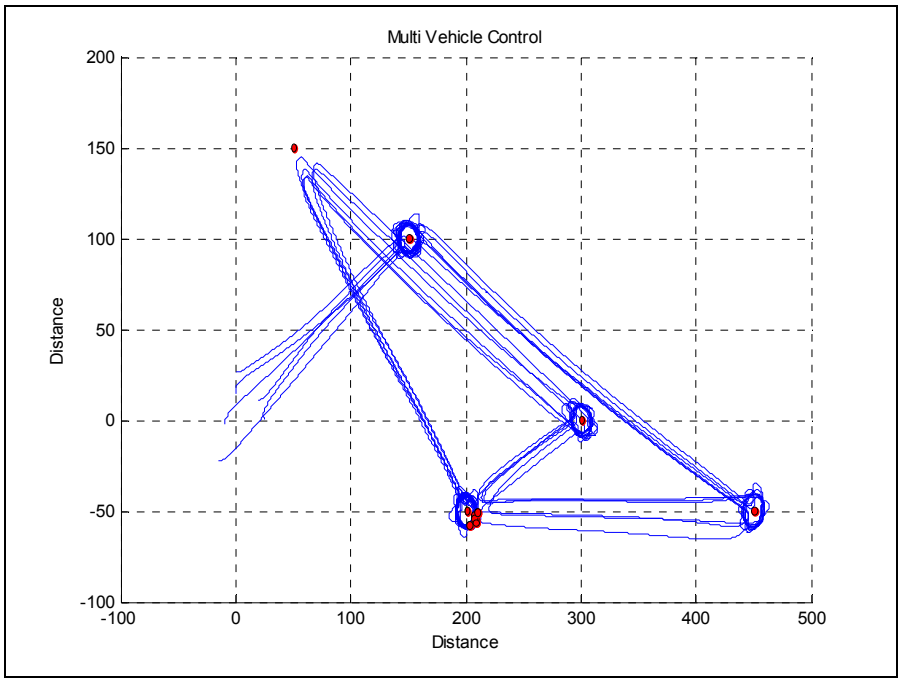


Figure 16: Swarm Response with Loiter Modification

7. Migration and Implementation of the NRL Multi-Vehicle Simulator

7.1. Software Acquisition

One deliverable for this project was the migration of NRL's multi-vehicle simulator to the Systems Department at USNA for both student and faculty use in future research. The simulator is a powerful tool that can be used to test control algorithms using vehicle models with advanced and accurate kinematic models. To facilitate this process, a step-by-step installation guide was developed (Appendix 4) for anyone to follow if they wish to set up a workstation with the multi-vehicle simulator which contains more detail about the process.

Specifically, several pieces of software were downloaded in order to acquire the simulator source code. The code is stored in a central location at NRL and using a process called subversion, multiple users are able to download copies of the code, modify the code, and upload any changes that may be beneficial to the group. All changes are stored on the central computer and the source code can be reverted to early version on a user by user basis.

In order to work on the NRL code, a JAVA compiler and editor called Eclipse was downloaded. The NRL multi-vehicle simulator also uses a CybelePro interface which supports agent to agent message transfer that is helpful in applications such as swarm simulation. CybelePro also limits the amount of information the UAV agents have access to by only allowing their position to be broadcast on the message traffic at certain times. This resembles the real world since no vehicle has real-time omnipotent knowledge of the properties of the other UAV and convoy agents. Once CybelePro is installed, the user is free to run simulations.

7.2. SIMDIS Support

Another role for this project was to serve as a link between the research at USNA and at NRL. To support this link, I have acted as the POC for questions concerning the use of NRL's

visualization software, SIMDIS. This tool allows USNA faculty and students to visualize their data in three dimensions. To facilitate the use of SIMDIS, a MATLAB script was created that is able to take a raw data file consisting of time and position data and convert it into the .asi file format that is needed by SIMDIS (Appendix 5). This allows students and faculty to use simulated or experimental data to create professional video files of their trials in a graphic rich environment for use in presentations or post data analysis. A user's guide was also created to allow students or faculty unfamiliar with SIMDIS to utilize the software without extensive knowledge or experience. This guide covers topics including creating presentation videos, plotting waypoints or desired trajectories, and visualizing sensor ranges (Appendix 5)

7.3. GPS Track Injection

Next, it was necessary to create a JAVA class that could read in a GPS text file and inject it into the simulator as convoy route. To get the GPS data, a Garmin handheld GPS receiver from NRL was used to log the vehicle's position every second. A program called GPSBabel was used to extract the GPS waypoints that were logged by the GPS and convert it to a form that was more convenient to work with. The final GPS track was saved in the form of a comma separated value file which followed the pattern of "waypoint number, latitude in degrees, longitude in degrees, altitude in feet, date, and time." This data is extracted and converted from degrees to radians for the simulator. Also the time stamps are converted from raw time to elapsed time.

The next step in the injection process is to broadcast this message to the other vehicles in the simulation. The vehicles update their position every second which is limited by the logging rate of the GPS receiver. To solve this issue, a timer is started when the simulation begins and every time the elapsed time of the GPS waypoint equals the elapsed time of the simulation, the GPS point is broadcast on a message channel using the CybelePro infrastructure. Also, the code

was modified to support time stamps down to the millisecond, which was artificially created through MATLAB scripts for use in several of the scenarios. This code was created and debugged with the assistance of a software engineer at NRL.

8. Mechanized Convoy Scenario - Rectilinear Control Investigation

The goal of this research was to find the set of gain values which provided the best performance for the urban convoy scenario. To begin this research, only the rectilinear form of the control algorithm was used and along with a GPS track that moved continuously at a constant speed with no stops. The goal of this scenario was to establish a testing procedure that could be applied to the other scenarios as well as find the set of parameter values that keep the vehicles close to the convoy but at a safe distance from each other while minimizing steering energy.

8.1. Data Analysis Plan

In order to find the optimal set of parameter values for the Mechanized Convoy, Foot Patrol, and Obstacles en Route scenarios, it was necessary to create a separate data analysis script file to assimilate the data from each of the simulation runs and condense the information into easily readable plots and tables. The majority of the data analysis was done by computing the averages for the simulation runs, identifying the parameter values which repeatedly had the best performance values, and locating any trends in the data by purely visual means. Thus, the foundation of the data analysis was drawn from a cost function analysis. In order to get the initial performance values from each simulation run, a MATLAB script was used to import the data from the simulator and compute the performance metrics, print them to a text file, and produce plots of the data (Appendix 6.1).

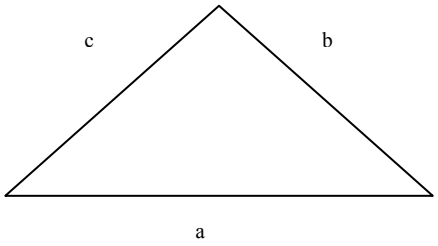
In order to determine an overall performance value for each parameter set, a cost function was used to combine the three main performance metrics into one value. First, the distance between vehicles is calculated over the entire simulation run. The percentage of the time that the intra-vehicle distance is below a user specified minimum value is reported for this performance metric. This is a binary metric, meaning that the vehicle is either above the minimum distance or below it.

The second metric is the distance of the vehicle away from the convoy. The performance metric represented the percentage of the time that the vehicles were outside of a maximum distance away from the convoy and was calculated in a linear method and not a binary method as the collision metric was. When the vehicle is at or under a minimum distance, there is no penalty. However, as the vehicle moves away from this minimum distance, the performance penalty increases linearly until it reaches a value of one. At a value of one, the vehicle is at or beyond a specified maximum distance from the convoy. This way the performance metric is a strong representation of the distance between the UAVs and the convoy and penalized larger values of r_0.

Lastly, it was necessary to calculate a performance metric to determine the fuel efficiency of the UAVs as well as a method of penalizing large gain values. As the gains increase, they create larger and larger values of *u*, the output of the algorithm. However, due to the vehicle kinematics, there are physical limitations on the size of *u*. For example, a UAV can not make a 90 degree turn; they have a certain turn radius which limits the steering control. To account for this, *u* is set as to not exceed some specified value in the simulations. As the gains increase, *u* increases until it is hitting its maximum value. At these points where *u* is saturated, the control law desires the vehicle to turn at an angle more than allowed by the dynamics of the vehicle.

This is called a slew rate limited condition and is a control problem. Since the vehicle is not physically capable of turning in the desired direction, the control law loses effectiveness and there exists a greater chance of collision with the other vehicles.

To prevent this case and to put a penalty on larger gain values, it was necessary to compute the average steering command *u* for the vehicles and compare it to the maximum value. This would result in a percentage of the amount of steering control being calculated out of the maximum allowed value and also serve as a gauge to determine the relative fuel efficiency of the system. The more the vehicles turn, the more fuel they burn. Therefore, smaller values of steering are desired. To compute this value, Heron's formula was used which states that for any three points in a trajectory, the amount of curvature can be calculated as an inverse of the radius of the unique circle through those three points. The exact formula is reproduced in Equation 2.



$$s = \frac{a+b+c}{2}$$

$$|k| = 4 * \frac{\sqrt{s(s-a)(s-b)(s-c)}}{abc} \qquad [2]$$

- a, b, and c are the length of the sides of the triangle.

By calculating the curvature this way, it is possible to get a good estimate of the steering command *u* that is being calculated by the control algorithm. This calculation was done in the post processing of the data instead of extracting the control value from the simulator for several reasons. First, this specific performance metric was not added until after analyzing the first

complete set of data for the first scenario. It was added to penalize high parameter values and to gain some insight on fuel consumption. Primarily, the simulator was coded to export vehicle position and time data only. Due to the complex nature of the program, adding additional code to capture the correct value of *u* would have been a time intensive process with limited JAVA experience. Instead, it was decided to estimate *u* and include the calculations in the MATLAB script that contained all of the other performance metric calculations. Lastly, for the purposes of this metric, the exact value is not required. This metric shows how close the control law is to maxing out the dynamics of the vehicle. As long as the same calculation is done on every vehicle, the results can be analyzed effectively.

Next, a script was written which would combine these three separate performance metrics into one single value (Appendix 6.2). This was done using a weighted average of the three values. The sum of the values added to 1, and the values ranged from 0.1 to0 .8 to create 36 permutations. In order to identify which parameters had specific effects on specific performance metrics, a process of varying the weights for each metric was used. Each weight was varied from 0.1 to 0.8 such that all of the weights added up to 1 (Appendix 6.3). A final performance value for each parameter set was created by summing the products of the individual weights and performance metrics. After each set of parameters had a computed performance value, these values could be combined based on initial conditions, route, number of vehicles, or even the entire scenario. This way, it is simple to compare the results of the simulations as an average for each set of parameters and determine the relationships, if any, to the performance metrics being considered.

8.2. Test Matrix Development

To begin the investigation, it was necessary to develop a test matrix which contained all the values of the parameters that would make the initial search space. This consisted of locating minimum and maximum values for all four parameters: α, μ, η, and r_0. Dr. Justh had done some previous research to develop a basic mathematical formula used to calculate the maximum r_0 value based on the number of vehicles in the swarm (Equation 3)

$$r_o = \frac{r_{sep}}{2}\sqrt{\frac{n-1}{2}} \qquad \textbf{[3]}$$

where r_{sep} is the separation distance between vehicles when circling and *n* is the number of vehicles including the waypoint or convoy in this case. It was then suggested to make the minimum values a factor of ten below the maximums. For the separation distance r_{sep}, the vehicle dynamics were used to find a minimum value. Since the GPS waypoint is updated only once per second, the minimum separation distance of 50 length units was used since the maximum speed of the UAV models is roughly 40 length units per second. This resulted in a range of 0.05 to 0.5 for α, η and μ and a range of 50 to 1650 for r_0. These were interpolated to produce four values per parameter. α, η and μ had the vales of 0.05, 0.15, 0.35, 0.5 and r_0 went from 50, 550, 1100, 1650. Thus each parameter had four different values resulting in 256 permutations in the test matrix for the rectilinear control law investigation.

8.3. GPS Track Data Manipulation

In order to get practical and meaningful results, the control algorithm had to be tested under a variety of GPS routes, initial starting positions, and with a varying number of vehicles in the swarm. For the routes, two different paths were logged. The first route was a GPS track taken

from a suburb of Washington D.C. to NRL in Anacostia. The second track was a route from NRL to USNA. In order to accurately analyze the effects of the different parameters on the swarm's behavior, it was necessary to modify the raw GPS data. For this first scenario, basic simulations were run with the goal of testing the pure rectilinear form of the control law. This meant that the GPS vehicle needed to move at a relatively constant and steady speed. However, when these tracks were logged with the GPS in a vehicle, traffic, stoplights and other obstructions forced the vehicle to stop or slow down significantly. To solve these issues, a MATLAB script was created that would read the raw GPS data file and filter out any of these areas where the vehicle was stopped. This process is explained in detail in Appendix 7. This code was slightly modified to create timestamps for the latitude and longitude positions that make the convoy travel at a constant speed. This was done by first calculating the distance between two points and then dividing by a speed such as 40 mph, to get a time between the two points. This time is added to the current time to get the next waypoint timestamp adjusted according to a specific speed. The result is a GPS track that has a constant speed and no loitering points.

8.4. Initial Conditions

For the first scenario, two separate routes were used. Refer to figures 17 for Route 3 and Figure 18 for Route 4.

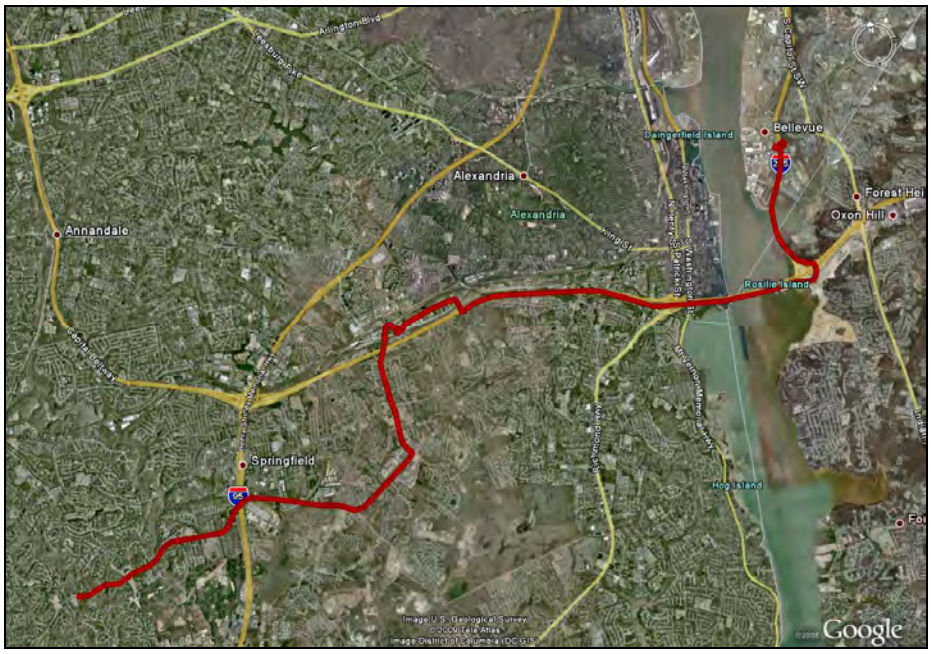


Figure 17: Suburb to NRL

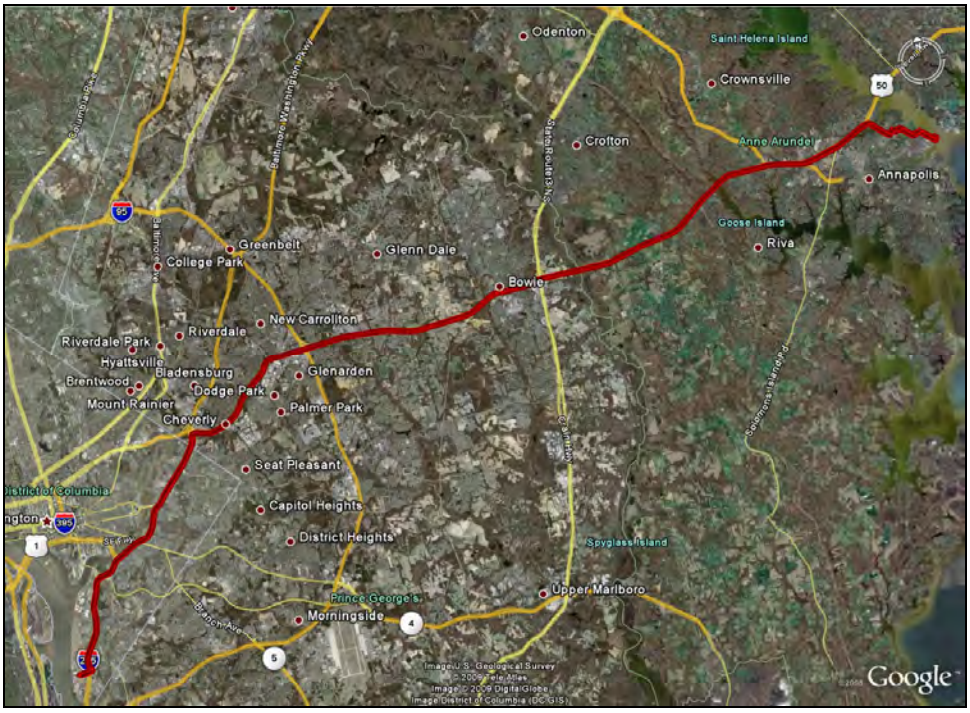


Figure 18: NRL to USNA

From these routes, initial conditions were chosen. In order to prevent the occurrence of discovering one set of parameters that works very well for only one specific case, it was necessary to test the parameters over a variety of initial conditions. This way, an average performance over the starting locations could be taken to improve the accuracy of any results from data analysis. The set of initial condition is documented in the table in Figure 19.

Scenario 1 Initial Conditions Table

Name	Scenario	Route	Convoy		UAV 1		UAV 2		UAV 3	
IP1	Close behind	1	38.75717	-77.2313	38.75882	-77.233579	38.75545	-77.233471		
IP2	Close front	1	38.75717	-77.2313	38.76351	-77.22445	38.753329	-77.222144		
IP3	1 close, 1 far	1	38.75717	-77.2313	38.75326	-77.229722	38.779234	-77.223464		
IP4	All far	1	38.75717	-77.2313	38.77923	-77.223464	38.737324	-77.2437691		
IP5	Close behind	2	38.82176	-77.0227	38.8229	-77.024016	38.82045	-77.024247		
IP6	Close front	2	38.82176	-77.0227	38.82396	-77.02093	38.820183	-77.02035		
IP7	1 close, 1 far	2	38.82176	-77.0227	38.8202	-77.02247	33.8301	-77.0252		
IP8	All far	2	38.82176	-77.0227	38.8157	-77.01412	38.83048	-77.022576		
IP9	Close behind	1	38.75717	-77.2313	38.75545	-77.233471	38.75882	-77.233579	38.757	-77.2365
IP10	Close front	1	38.75717	-77.2313	38.75176	-77.21756	38.76977	-77.2187016	38.75746	-77.226967
IP11	1 close, 1 far	1	38.75717	-77.2313	38.74915	-77.2313	38.77861	-77.20861	38.75668	-77.243638
IP12	All far	1	38.75717	-77.2313	38.73432	-77.2069	38.77861	-77.20861	38.7549	-77.27418
IP13	Close behind	2	38.82176	-77.0227	38.8229	-77.024016	38.82045	-77.024247	38.8216	-77.025739
IP14	Close front	2	38.82176	-77.0227	38.82396	-77.02093	38.820183	-77.02035	38.82191	-77.01871
IP15	1 close, 1 far	2	38.82176	-77.0227	38.8202	-77.02247	33.8301	-77.0252	38.7565	-77.2335
IP16	All far	2	38.82176	-77.0227	38.8157	-77.01412	38.83048	-77.022576	38.82098	-77.03659

Figure 19: Scenario 1 Vehicle Starting Locations

8.5. Data Analysis of the Simulation Results

Since there is a limitless supply of data possible for this project, it was necessary to decide what type of conclusions would be drawn from the data. This would limit the time spent on a specific scenario and allow time for the investigation of the other scenarios by directing the focus of the analysis. The following conclusions were deemed the most appropriate for the goals of this project.

8.5.1. r_0 and Collision Avoidance

From the data averaged over all sixteen trials, it was clear that having a separation distance equal to the minimum vehicle separation distance used to determine the collision metric

greatly decreased the swarm's performance. While several factors such as convoy speed and direction play a role in the intra-vehicle separation, setting a vehicle separation distance equal to the collision distance results in a large increase in chance of collisions, as expected. However, if the separation distance is increased to twice the minimum collision distance, the risk of collisions stays low and in most cases is zero. This parameter was the largest contributor of trials with poor performance because of the significant increase in collision percentage when r_0 was roughly equal to $r_{\min}$ (the minimum range between UAVs at which collisions occur). As r_0 continued to become greater than $r_{\min}$, the collision percentage dropped to 0 in the majority of cases. However, once r_0 became roughly twice $r_{\min}$, the r_0 parameter played a less noticeable role in overall performance of the swarm.

8.5.2. r_0 and Convoy Range

One of the unexpected results from the data was the trend that the minimum value of r_0 did not result in the lowest convoy range percentages. On average, the parameter sets with r_0 equal to 550 had slightly lower convoy range percentages than the sets when r_0 was equal to 50. However, when r_0 was greater than 550, convoy range did increase as expected. The one clear difference between the parameter sets with r_0 equal to 50 and the ones with r_0 equal to 550 was the collision percentage. Chances of collision were much higher when r_0 was equal to 50. This may have caused the sets with r_0 equal to 550 to have a slightly lower convoy range percentage. These vehicles were spaced far enough that they did not need to make any drastic maneuvers to avoid collision and were thus able to focus more on their convoy separation distance. When r_0 was 50 and chance of collision was high, the vehicles had to spend more energy not crashing which resulted in the vehicles not staying as close to the convoy as expected. Also, as discussed

earlier, the control law is a sum of behaviors which does not result in precise and predictable behavior. While the user may want the vehicles to be very close to the convoy, it does not appear possible that the rectilinear form of the control law is able to support these desires. It appears that there is a limitation on how close the vehicles are able to stay to each other while avoiding collisions.

8.5.3. Steering Energy Analysis

Also of note was the apparent relationship between the separation distance r_0 and the amount of steering energy used during the simulation. As the separation distance increased, there was a noticeable drop in the amount of steering energy used once r_0 was no longer equal to the minimum collision distance. The vehicles expended much more energy when r_0 was 50 trying to avoid collisions. As the distance between the vehicles decreases, the variable r_{jk} in the control law is also decreased causing the output of the control law, *u*, to grow. As the control law demands steering commands with smaller magnitudes, the amount of steering energy needed is also decreased. When r_0 was increased and the chance of collisions was minimal, steering energy remained more consistent as r_0 increased.

After evaluating the data, it is evident that larger values of the parameters led to larger values of steering energy. Specifically, high values of μ and α caused much larger values of steering energy with relatively lower values of η. Thus, when the control law is weighted so that the vehicles are strongly driven perpendicular to a baseline (η is high) while their heading and spacing behaviors are relatively weak, the resultant behavior utilizes small amounts of steering energy. However, when μ and α are given larger values and the control law weights heading and separation behavior high, the resultant behavior uses much more steering energy to ensure

that these behaviors are met. Thus, if direction and vehicle separation are important behaviors for a specific scenario, the result is a decrease in fuel efficiency as seen by the use of more steering energy.

8.5.4. Sensitivity of the Minimizing Parameter Set

One of the conclusions drawn from this analysis was the apparent sensitivity of the global minimum value with respect to the weights of the performance metric. It was necessary to determine how the global minimum changed if the weights of the final cost function were adjusted to place varying levels of emphasis on the individual metrics. This would show which parameter sets performed better when different performance metrics were emphasized. It also showed how well the global minimum parameter sets responded to changes in cost function weighting. To determine this, a script file was used which located the global minimum parameter set while varying the weights of the cost function. (Appendix 6.3)

From this data collected across all 16 initial conditions, there were two parameter sets which were global minimums. Parameter set 21 (μ=0.2, α=0.2, η=0.05, r_0=50) appeared eight times and parameter set 85 (μ=0.2, α=0.2, η=0.05, r_0=550) appeared 28 times. Performance wise, parameter set 21 had lower convoy range percentage but had a very high collision percentage. Parameter set 85 had a higher convoy range percentage but a zero collision percentage. Both parameter sets had similar steering energy percentages. Parameter set 85 was the global minimum for the majority of the cost function weights and set 21 only appeared when collision was weighted very low and convoy range was weighted higher. Since only two trials appeared as the global minimums as the cost function weights were manipulated, it can be concluded that the parameter sets are not sensitive to the weighting of the performance metric. The same few trials repeatedly outperform the other trials as the weights are changed and

continue to result in strong performance. This shows that these two parameter sets are not only the top sets for certain situations, but they create behavior that is ideal across all of the performance metrics. Parameter set 85 would be a very strong choice of parameter values for use in a purely rectilinear scenario.

Also of note were the parameter values that the global minimum trials were comprised of. For these parameter sets, μ was 0.2, α was 0.2, η was 0.05 and r_0 was 50 or 550. Clearly, these gain values yielded very good swarm performance and the controller is very sensitive to the values of the gains. Gain values were kept low which resulted in low steering energy performance. In general, when the highest weight was given to α and μ equally and the lowest weight given to η, the best performance was observed. However, because η was always .05 for optimal trials, there were no parameter sets which contained every value strictly inside the test matrix. Perpendicular alignment to the baseline has a very small influence on overall swarm performance but the ability for the vehicles to maintain a constant separation distance is critical to increasing the performance of the swarm. The η term probably has a larger influence in the beginning of the simulations, moving the vehicles from initial conditions to steady state. However, as the vehicles reach this steady state, they rely more on α and μ for improved performance.

While parameter set 85 had a high convoy range percentage relative to set 21, it must be taken into account that it had a higher r_0. Parameter set 85 had an r_0 of 550 and a convoy range of 52% which corresponds to an average distance of about 800 distance units. Parameter set 85 was able to maintain an average distance much closer to its r_0 distance compared to parameter set 21. The discrepancies between the convoy range percentage and r_0 may be due to the collision percentage as discussed earlier, or a relative speed difference between the convoy and

the vehicles. There may be some portions of the convoy track where the vehicles simply can't keep up with the convoy and fall out of range. It should also be noted that while r_0 changed for these two parameter sets, the values of their gains did not change at all. It seems that these specific values offer ideal performance regardless of the separation distance. The gain values and r_0 are not dependent on each other. The gain values offer the best performance and the r_0 value dictates the convoy range, collision risk, and steering energy to be used.

8.5.5. Performance Around the Global Minimum

In order to get a stronger idea about the behavior of the swarm with respect to each of the parameters, one parameter at a time was varied around the global minimum and the effects on each of the metrics and the overall parameter performance was observed. Parameter set 85 was chosen since it appeared most often as the top performing parameter set. Parameter set 85 had a collision percentage of zero which made analysis of the parameter effects on collision avoidance impossible. In order to examine the effects of the parameters on collision avoidance, set 21 was used.

As μ became larger or smaller than 0.2, the convoy range percentage increased as well the overall performance. The goal was to minimize the performance percentages; therefore an increase in the performance percentages resulted in degraded swarm behavior. When μ was less than or equal to η, steering energy was low. However, when μ was given the highest weight in the set, steering energy increased. This shows that as a greater emphasis is placed on heading alignment, more steering energy is needed. When μ was 0.35 or less, collision percentage was unaffected. Only when μ was given the maximum value of 0.5 was there a significant decrease in collision percentage. This shows that as the heading alignment term is given a stronger weighting, the chance of collisions decreases.

As α became larger or smaller than 0.2, the convoy range percentage and overall performance values increased. As α increased from 0.05 to 0.5, the risk of collision decreased while steering energy increased. As α gets a higher weight, there is a smaller chance of collisions and more steering energy is used. The increase in steering energy could be a result of both the desire of the swarm to maintain a certain distance as well as preventing collisions. The increase in steering energy due to α is not as large as the increase due to η. This means more steering energy is used aligning heading than maintaining proper separation distance.

As η decreased from the value of 0.5 to the minimum value of 0.05, convoy range percentage and overall performance values decreased. As less emphasis was put on the perpendicular baseline alignment, the vehicles were able to maintain closer distances to the convoy. If this behavior is minimized by having a low parameter value, the vehicles are allowed a greater range of motion. The swarm focuses on separation distance and heading rather than being aligned with the convoy. This behavior allows the swarm to stay at a closer distance to the convoy. Steering energy remained relatively unaffected while η was 0.05 to 0.35 and only increased slightly when η was given the maximum value of 0.5. This shows that the η behavior is not a large factor in determining the movements of the vehicles. Once the vehicles are in a steady state formation, the α and μ parameters have much more control over the maneuvering of the vehicles. There was no clear trend in the data for collision avoidance. It seems that this behavior has little control over this performance metric when compared with the other parameters.

Lastly, variations with respect to r_0 were analyzed and trends similar to those mentioned earlier were discovered. As r_0 increased, the collision avoidance dropped drastically after $r_0 > r_{\min}$ and steering energy dropped as well. Convoy range percentage increased as r_0 increased,

which is expected. As one moved away from an r_0 of 550 in either direction, overall performance percentage increased. It was also shown that once r_0 is greater than $r_{\min}$ and the collision percentage drops to zero, steering energy remains roughly the same. This shows that as the size of the orbit increases, the increase in the amount of steering energy is not significant. The α and μ parameters have a much greater effect on steering energy than r_0.

8.5.6. Comparison of 2 and 3 Vehicle Trials

One of the goals of this analysis was to determine if implementing a gain scheduling process that modified the gains of the control law based on certain physical conditions of the system was an efficient method to create consistent swarm performance. One case where this could be beneficial is if the swarm size were decreased either due to a malfunction or a crash of one of the vehicles. It may be possible that certain parameter sets offer better behavior for different size swarms, and thus, the user would want to switch between parameter sets as the swarm size changed.

When the performance averages for all trials where the swarm was 2 vehicles were analyzed across varying weights of the cost function, parameter sets 21, 41, 85, and 106 were identified as global minima. For these parameter sets, μ ranged from 0.2 to 0.35, α from 0.2 to 0.35, η ranged from 0.05 to 0.2 and r_0 ranged from 50 to 550. These gain values are slightly higher and have a larger range of values than the sets for all trials averaged together. This may be due to the fact that since the swarm size is low, performance can be enhanced by using higher gain values. There is less interaction between vehicles which results in less average steering energy being used and also a smaller risk of collision. This leads to the conclusion that as the swarm size increases, the values of the parameters are decreased. For all of the parameter sets, μ and α were always equal. Also, η was always the smallest value. η only increased from 0.05 to

0.2 when r_0 increased from 50 to 550 and α and μ increased from 0.2 to 0.35. The increase in r_0 had no effect on the α and μ parameters. Both parameter sets 21 and 85 appeared in both the 2 vehicle minimum parameter sets as well as the sets for all trials averaged together. Parameter sets 41 and 106 were slightly more aggressive versions of sets 21 and 85.

Analysis of the 3 vehicles trials revealed results more in line with the averages of all 16 trials together. Parameter sets 21 and 85 were global minimum as the cost function weights were varied with μ equal to 0.2, α equal to 0.2, η equal to 0.05, and r_0 equal to 550 and 50. Both parameter set 21 and 85 appeared in the 2 vehicle global minimum sets. All of the gains become more sensitive when another vehicle is added. The addition of the extra vehicle resulted in a decrease in the magnitude of the gains and an increase in r_0. This results in a decrease in collision percentage when compared to the results from the 2 vehicle data sets. This is due to the fact that the separation distance is always 550 in the 3 vehicle case and split between 50 and 550 in the 2 vehicle case. The vehicles are farther apart for the 3 vehicle minima which have a stronger influence over the collision behavior than the value of the gains. This results in fewer collisions for the global minima parameter sets, even though there are more vehicles. While the addition of a vehicle may result in fewer collisions, it also drastically increases the average range from the convoy and increases the average steering energy as well. The paths of the vehicles are less flexible as more vehicles are added and in order to prevent collisions, range to the convoy is sacrificed.

When the top performing parameter sets for 3 vehicles are evaluated in the 2 vehicle conditions, they still perform well with set 85 in the top 5 parameter sets and set 21 in the top 25 depending on the cost function weights. However, when the 2 vehicle parameter sets are analyzed for 3 vehicle scenarios, they perform poorly, roughly in the middle of the test matrix.

Only parameter set 85 performs consistently well regardless of the number of vehicles in the swarm. As the size of the swarm increases, the parameter sets begin to match those of the top parameter sets for all of the trials. While there may be a slight increase in performance by switching from the top parameter set to one specific for 2 vehicles, keeping the original parameter set for 3 vehicles will result in very similar performance. Thus, it is not clear that gain scheduling would be necessary for the case where the swarm is reduced in size from 3 to 2 vehicles based on this data. However, if the swarm is initially at 2 vehicles and parameter values set for 2 vehicles are used and the swarm size increases, it would be beneficial to change the parameter values to a set with better performance with a 3 vehicle sized swarm.

While moving from 2 to 3 vehicles results in some minor changes, what will happen as more vehicles are added to the swarm? From this analysis, the trend appears that the r_0 value would increase to the largest value possible while the magnitude of the gains would decrease. As vehicles are added, the parameter sets would approach the same values as those found in the global minimum of the largest swarm size. Thus, if one took the global minimum for the largest swarm and used those same values for swarms of smaller size, there should not be any significant loss in performance. However, if the global minimum for a small vehicle case is applied to a large swarm, there would be a significant and detrimental decrease in performance.

8.5.7. Comparison of Route 1 and Route 2

It was also necessary to compare the performance of the swarm over the different routes that were used. If there were clear preferences for certain parameter sets based on the physical attributes of the route, it is possible that certain gains could be selected based on these properties. By examining the two routes, it is clear that Route 2 followed a straighter path and had less extreme turns when compared to Route 1. Therefore, Route 1 caused the swarm to maneuver

more which would lead to more collisions, more steering energy and possibly a further convoy range percentage. The minimum parameter sets for the two routes were found as the performance weights of the cost function were changed.

Route 1 had parameter set 85 as a minimum all 36 times. μ equaled 0.2, α was 0.2, η was 0.05 and r_0 was 550. This showed that the performance of all the parameter sets in relation to Route 1 was extremely sensitive. For all cost function weighting, parameter set 85 performed the best. This also means that parameter set 85 had the best performance in all three performance metrics. It may be possible that this parameter set is highly flexible in the formations that it creates which allows the swarm to be responsive to the changes on convoy direction.

Route 2 contained global minima parameter sets with more variability. Parameter set 1 appeared once, set 21 appeared eight times, set 41 appeared once, set 85 appeared twenty times and parameter set 93 appeared 6 times. Here, μ ranged from 0.05 to 0.35, α from 0.05 to 0.35, η was 0.05 and r_0 ranged from 50 to 550. When the route requires less maneuvering, the control law is able to use a wider variety of gain values. These sets also reflected the observation that the best performance occurs when α and μ are equal and η is small. When r_0 was 50, the μ and α values ranged from 0.05 to 0.35. However, when r_0 was 550, α and μ were 0.2. This shows that when the vehicles are required to stay closer to each other and the convoy with a small r_0 value, which increases the risk of collisions, the parameter sets become relaxed to support a wider range of swarm behaviors. This wider range allows the swarm to remain close to the convoy while reducing the chances of collision along with steering control.

Only parameter set 85 appeared in both the Route 1 and Route 2 global minima. When all sixteen trials are averaged together, set 85 is the global minimum a majority of times, but the appearance of parameter set 21 may be due to its performance when the route is more direct.

Parameter set 85, a global minimum for both routes, appeared when the convoy range percentage weight was low and the steering performance was emphasized. The convoy range percentage was around 46% for Route 1 and jumped to about 57% for Route 2 while steering energy remained roughly the same. This prevented parameter set 85 from being the global minimum for more sets in Route 2.

It appears that as the route becomes less complex, parameter set 85 actually has more trouble following the convoy than when the route requires more maneuvering. This contrasts with the other parameter sets. For sets 1, 21, 41, and 93 the convoy range percentage decreases from Route 1 to Route 2. This is logical since Route 2 requires less maneuvering. This allows the vehicles to get closer to the convoy without a large risk of collisions. Since the convoy path is fairly linear, the vehicles can move into a steady state position and remain in that position without much readjustment. This reduces the chance for collision and the overall steering energy as well. While parameter set 85 improves performance when the route is more complex, the majority of the other parameter sets have better performance when the route is more linear. Parameter set 85 was able to keep the vehicles close to the convoy with minimal steering energy and without the high chance of collisions that are common with parameter sets with a r_0 equal to 50. The ability of parameter set 85 to adapt to a variety of routes is a primary reason that it appeared as a global minimum when all of the trials were averaged together.

When comparing the performance metrics of the global minima for the two routes, there are several trends. First, the average range to the convoy is higher for Route 2 when compared to Route 1. By using a more linear route, relative speed differences are exaggerated, resulting in an increased convoy range in Route 2. Since for both routes, the steering percentages are roughly the same, the vehicles are not maneuvering more to respond to the convoy. If there is a large

speed difference but the convoy maneuvers repeatedly, the vehicles are given time to catch up. However, if the convoy follows a straight path, the vehicles have no chance to catch up and continue to fall behind the convoy.

On average, collision slightly decreased for Route 2 when compared to Route 1. In Route 1, the vehicles are forced to maneuver more to stay headed in the same direction as the convoy, resulting in an increased use of steering energy and leading to a higher risk for collision. If the convoy path was straight in one direction, it would be expected that the steering energy would go to 0 once all the vehicles had reached steady state positions and headings and therefore, the risk of collisions would also go to 0. This seems to be similar to what happens in Route 2. However, as the convoy is constantly changing direction in Route 1, the swarm is forced to respond. Thus, Route 2's lower level of complexity reduces collisions.

When comparing the parameter ranges for the global minima for the two routes, it is evident that these ranges closely mimic those of the parameter sets that are local minima when all 16 trials are averaged together. μ remains 0.2 for Route 1 and ranges from 0.05 to 0.35 in Route 2. However, when r_0 is 550 for Route 2, μ remains 0.2, which is the same when all trials are averaged together. This contrasts with the analysis of the two versus three vehicle scenarios where μ ranged from 0.2 to 0.35. It appears that μ is more sensitive to the number of vehicles rather than the properties of the route of the convoy when r_0 is 550. This could be due to the fact that as more vehicles are added, there are more headings to align, and a greater weight is needed on this behavior to produce effective performance from the swarm.

Lastly, the magnitude of r_0 was varied between the routes. Where Route 1 had every parameter set with a r_0 equal to 550, Route 2 had r_0 values split between 50 and 550. Since Route 1 has more maneuvering, a higher separation distance was needed to prevent collisions

and minimize steering energy as much as possible. This contrasted with Route 2 where the vehicles could afford to be closer together with a smaller r_0 since there were less turns and thus fewer opportunities for collisions.

Overall, Route 2 was able to use a wider range of gain values and smaller r_0 when compared to the parameter values for Route 1. The flexibility of gain values allowed the controller to decrease steering energy while bringing the vehicles closer to the convoy with a smaller r_0. This was done at the expense of possibly increasing collisions between vehicles. This was possible since Route 2 was less complex compared to Route 1 and the chance of collisions was less likely.

9. Foot Patrol Scenario – Circular Control Investigation

The next scenario for analysis was designed to test the parameters of the circling form of the control law. To transition to this form of the algorithm the heading alignment term μ is dropped. If the user desires the vehicles to maintain a circular orbit, the vehicles should not have their headings aligned. This prevents problems when vehicles are on opposite sides of the orbit and facing opposite directions. By removing the alignment behavior, the control law is able to support stable circling formations which are useful when tracking slow moving convoys or foot patrols. Thus, convoy routes that mimicked foot patrols were created in order to isolate the circular control law for analysis.

9.1. Convoy Trajectory

For this scenario, the convoy needed move at a slow and steady pace simulating a foot patrol. Having obstacles in the route of the convoy was no longer an issue. The convoy would be moving at such a slow speed that obstacles would not have an affect on the swarm's

performance. To get the GPS data, a path walked around USNA was recorded and is shown in Figure 20. Next, a route take at a running pace was logged in downtown Annapolis and is shown in Figure 21.

Figure 20: USNA Foot Patrol

Figure 21: Annapolis Foot Patrol

Specifically, a goal of this scenario was to determine at which convoy speed the circling form of the control law began to lose stability. This was tested by using foot patrols at two different speeds. This knowledge would prove useful when attempting to create a blending algorithm based off of the convoy's speed. It was also necessary to add a five minute delay to give the vehicles time to maneuver into a stable orbit before the convoy started moving. This resulted in the convoy remaining stationary for 5 minutes and then beginning the route. The code used for this modification of the GPS tracks can be seen in Appendix 7.

9.2.Initial Conditions

Similar to Scenario 1, this Scenario had a breakdown between 2 and 3 vehicles, the two routes discussed above, and the four sets of initial starting locations for the vehicles to produce 16 total trials. The starting locations of all the vehicles are depicted in the table in Figure 22.

Scenario 1 Initial Conditions Table										
Name	Scenario	Route	Convoy		UAV 1		UAV 2		UAV 3	
IP1	Close behind	3	38.9810478	-76.48075906	38.9802054	-76.4860928	38.9834205	-76.48506		
IP2	Close front	3	38.9810478	-76.48075906	38.9821029	-76.4756455	38.9771306	-76.4824217		
IP3	1 close, 1 far	3	38.9810478	-76.48075906	38.9802054	-76.4860928	38.9932055	-76.4870563		
IP4	All far	3	38.9810478	-76.48075906	38.9932055	-76.4870563	38.9780114	-76.4972665		
IP5	Close behind	4	38.97861617	-76.48467523	38.981971	-76.4880643	38.9769139	-76.4797995		
IP6	Close front	4	38.97861617	-76.48467523	38.9747008	-76.4830384	38.9792478	-76.4899496		
IP7	1 close, 1 far	4	38.97861617	-76.48467523	38.9818528	-76.4812726	38.9918707	-76.4968082		
IP8	All far	4	38.97861617	-76.48467523	38.9918707	-76.4968082	38.9735385	-76.4631502		
IP9	Close behind	3	38.9810478	-76.48075906	38.9802054	-76.4860928	38.9834205	-76.48506	38.9850818	-76.4798841
IP10	Close front	3	38.9810478	-76.48075906	38.9821029	-76.4756455	38.9771306	-76.4824217	38.9788254	-76.4761842
IP11	1 close, 2 far	3	38.9810478	-76.48075906	38.9802054	-76.4860928	38.9932055	-76.4870563	38.9821029	-76.44756455
IP12	All far	3	38.9810478	-76.48075906	38.9932055	-76.4870563	38.9780114	-76.4972665	38.9745231	-76.4657372
IP13	Close behind	4	38.97861617	-76.48467523	38.981971	-76.4880643	38.9769139	-76.4797995	38.9818528	-76.4812726
IP14	Close front	4	38.97861617	-76.48467523	38.9747008	-76.4830384	38.9792478	-76.4899496	38.9756587	-76.4882937
IP15	1 close, 2 far	4	38.97861617	-76.48467523	38.9818528	-76.4812726	38.9918707	-76.4968082	38.9756587	-76.4882937
IP16	All far	4	38.97861617	-76.48467523	38.9918707	-76.4968082	38.9735385	-76.4631502	38.9648132	-76.4976047

Figure 22: Scenario 1 Initial Conditions

9.3.Data Analysis

Since the circling form of the control law did not contain the μ gain, the data analysis for this scenario was a simpler process and it was more clear which behaviors and performance metrics were affected by changes in specific parameters.

9.3.1. r_0 and Collision Avoidance

Similar to the trend observed in the rectilinear control law, as the r_0, or separation distance between the vehicles is decreased, there is an increase in the chance of collisions reflected by an increase in the collision percentage. Overall, the circling form of the control law established formations that had less chances of collision when compared to the rectilinear form of the control law. This can be explained by the types of formations that the control law forms produce. While neither form of the control law designates specific vehicles to occupy specific locations, they do control the shape of the swarm through the implementation and weighting of the behaviors that comprise the control law. For the rectilinear case, the resulting shape is a constantly evolving formation that adapts to the locations of the vehicles and the convoy. The circular form of the control law lends to swarm formations of a much more constant and stable

shape. This formation is much less sensitive to any abrupt direction changes of the convoy when compared to the rectilinear law. It also results in swarm behavior that once established, has zero chances for collision if the steady state formation is able to be kept. The only real chance of collision for this scenario occurs as the vehicles move from their initial positions into the steady state circling formation. While r_0 does have an affect on collision percentage due to the fact that the closer the vehicles are supposed to be, the more likely they are to collide, the behavior generated by the circling control law lends itself to collision free formations.

9.3.2. r_0 and Convoy Range

Also of note was the general relation between r_0 and the average distance to convoy percentage. Logically, as r_0 is decreased, the radius of the circle that is created by the circling form of the control law decreases, and the vehicles are able to remain in orbit at a closer distance to the convoy. This was reflected in the data as well. When r_0 was small, the vehicles maintained extremely close ranges to the convoy. As r_0, increased so did the average distance to the convoy. When r_0 was about equal to the maximum sensor range of 1 mile, average distances were about 98% in range. This meant that the vehicles were maintaining an orbit right at the limitations of the sensor. One observation from this scenario was the precision of the control law with respect to orbiting distances. At smaller values of r_0, the actual orbit was larger than the prescribed r_0. Only when r_0 was the largest value in the test matrix, did the actual orbit radii match with the prescribed r_0 value. The table in Figure 23 summarizes the differences between the prescribed r_0 value and the resulting average orbit radius as well as the percent error difference.

r_0 (yards)	Average Convoy Range Percentage (%)	Average Orbit Radius (yards)	Percent Error (%)
50	15.56234	273.8973	81.745
550	55.8735	983.373	44.07
1100	93.3624	1643.18	33.0566
1650	97.50476	1716.084	3.8508

Figure 23: Prescribed r_0 Values Compared to Experimental Values

When r_0 is low, there is a large margin of error r_0 and the actual separation distance calculated from the simulations. Also at a low r_0 level, there is a larger difference between the average convoy range values between the parameter sets. Some range as low as 6% while others are as high as 40%. However, as the size of r_0 is increased, the percent error between the r_0 parameter and the range to convoy percentage decreases until at an r_0 about equal to the maximum range to convoy, the percent error is effectively 0. Also as r_0 is increased, the variation between the parameters decreases. This means that maintaining a very tight orbit as prescribed by a low r_0 value is much more sensitive to the gain values than the case when r_0 is large. As r_0 is increased, the vehicles are given more room to maneuver with a larger orbit. Also, as the convoy moves, the relative distance between the convoy and the vehicles is much larger than the case when the vehicles keep a tight orbit. When a tight orbit is used, the vehicles must respond faster and in a more energy demanding manner to keep the tight orbit around the convoy and this increase in maneuver will rely on precise gain value. Thus, as r_0 increases, the average range to the convoy increases as expected, but the behavior is more representative of the prescribed r_0, more consistent, and less dependent on the gain parameters.

9.3.3. Steering Energy

As seen in the first scenario, as the values of the gains increase, the amount of steering control used increases as well. However, in the circling form of the control law, the steering energy was much more closely related to the value of α as opposed to η. When the data was sorted to locate those parameter sets with the lowest amounts of steering energy used, the best performing parameters sets all had very low values of α (0.05 or 0.2), but η ranged from 0.05 to 0.5 in the top 10 parameter sets. This is most likely due to the fact that α controls the size of the orbit that the vehicles form. While η may be important in getting the vehicles to the right positions and equally spaced around the orbit, it is the α parameter which determines how tight that orbit will be. This is directly related to steering energy since the tighter the orbit, the more the vehicles will have to turn and the more energy they will use. This data appears to support earlier conclusions that the η parameter is most effective when establishing the steady state formation of the vehicles and loses relevance once this position is generally established.

Also evident was a correlation between r_0 and the amount of steering energy used. As r_0 increases, the size of the orbit increases and the vehicles use less steering energy. This combined with the weight of α were the most influential factors that caused either very high or very low levels of steering energy to be used. Thus, to increase fuel efficiency, a user could lower α which would result in an orbit larger than specified by r_0, or increase r_0 directly. However, it should be noted that increasing r_0 has a much more predictable affect on the size of the orbit than decreasing α.

9.3.4. Sensitivity of the Minimizing Parameter Set

The script in Appendix 6.3 was used to compute the global minimum parameter set for all of the trials as was done before for the rectilinear control law. This produced parameter set 1 seven times, set 2 two times, set 6 thirteen times, and set 7 fourteen times. For these parameter sets, α ranged from 0.05 to 0.2, η ranged from 0.05 to 0.35 and r_0 was always 50. This was initially surprising since the smallest value of r_0 was consistently in the top performing parameter set. It would be expected that the gains achieved by creating a very tight orbit around the convoy and keeping the vehicles close would be offset by an increase in collision percentage and steering energy. As discussed previously, the separation distance only affects collision percentage as the vehicles are moving to their initial positions. In the top performing parameter sets, only set 7 had a non-zero collision percentage and even this percentage was less than 0.5%. It is evident that for the circular form of the control law, the initial conditions create the only cause for collision concern as vehicles transition into the steady state orbit.

These parameter sets also showed that swarm behavior was more closely related to r_0 while it was fairly insensitive to the sizes of α and μ. This shows that while steering energy may have increased for the smaller r_0 values, this increase was not as significant as the decrease in convoy range. By using a small r_0 and a small α, the swarm was able to stay very close to the convoy while still using relatively little steering energy. When r_0 was 50, even the largest convoy range was not as large as the smallest convoy range for the sets when r_0 was 550. However, certain values of α and μ used with an r_0 of 50 got values of steering energy comparable to the parameter sets with r_0 equal to 550. The results also reinforced the concept that as α decreased, steering energy decreased, but convoy range increased as a penalty. In

general, it was evident that the benefits of using a small r_0 value outweigh any steering penalty associated with the tighter orbit. It may be possible to make r_0 so small that collisions do occur and steering control is maxed out, but this is not practical. Having r_0 at the minimum collision distance proved that this parameter has no real impact on collision and provides the best overall performance.

None of the top performing parameter contained all three gain values strictly inside the test matrix. This was due to the fact that a small r_0 was the strongest indicator of swarm performance. This is not a large problem because researching parameter sets with an r_0 less than the collision distance is not practical. Any benefit in performance would not be worth an increased chance of collision. Parameter sets 6 and 7 had values of α and η that were both within the interior of the test matrix. These sets also made up the majority of the minima as the cost function was varied. These parameter sets had extremely low convoy range percentages as well as low collision and steering percentages. The performance for these two sets was about equal in all areas. The only case when these sets were not the global minimum was when steering control was explicitly emphasized, however, they were the top performers when all metrics were weighted evenly or preference was given to collision or convoy range. These sets also reiterated the trend that steering energy is most directly related to increases in α vice η. There were much larger jumps in steering energy as α increased than compared to increases in η. Overall, the circular law did not seem to be any more or less sensitive to cost function weighting than the rectilinear law.

9.3.5. Performance Around the Global Minimum

Since none of the top performing parameter sets contained all gain values within the test matrix, parameter sets 6 and 7 were used to evaluate the swarm performance around a global minimum since they contained values of α and η which were inside the test matrix. To analyze the swarm behavior on a per parameter basis, all parameters were held constant while only one was varied and the changes in the performance metrics were observed.

First, changes in α were observed. As witnessed before, steering energy increases in a fairly regularly pattern as α increases. Since r_0 for both of these parameter sets is 50, it would be expected that the vehicles remain close to the convoy. When α is at its lowest value at 0.05, convoy range is high at about 35%. When α increases to any larger value though, convoy range drops to about 8%. For both parameter sets, convoy range is smallest when α equals η. As discussed previously, if α is given a small weight compared to η, the resulting behavior does not create a stable orbit with a radius close to r_0. However, when α is increased to the same level of η, the behaviors are weighted equally, and the resulting orbit has a radius that is much more reflective of the r_0 specified. Also of note is the fact that once α and η are on the same orders of magnitude, increasing α has no further impact on the convoy range.

Similarly, if α is less than or equal to η, collision percentage is 0. Since there is no heading alignment term in the circular control law, η is the only behavior which prevents the vehicles from colliding. Since r_0 is about equal to the collision distance, α acts to increase the chance of collisions by bringing the vehicles closer to the collision distance and r_0. Therefore, when α is given a higher weight than η, the resulting behavior places a larger emphasis on separation distance and there are collision risks.

Next, the response as η is modified was observed. As shown in the α observations, the convoy range is minimal if α equals η. However, where α had a large impact on convoy range based on its relation to η, η seems to have a much smaller impact. As η moves, the differences in convoy percentages changes by only about 6% in either direction. Also observed was the relation between α, η and collisions distance. When η was given a larger value than α, collision percentage dropped to zero. However, when α was greater than or equal to η, collision percentage became a non zero number. Lastly, as η was changed, the values of steering energy varied by less than one percent. As seen earlier, steering energy is most directly correlated to the α gain and the size of η has no clear affect.

Lastly, the swarm behavior with respect to r_0 was observed. While it was not possible to asses the performance trends when r_0 decreased from the value of 50 which parameter sets 6 and 7 contained, it was not necessary since these values would not be practical in a real world application. Again it was observed that as r_0 increases, the convoy range increases as expected. Also, it was noted that as the orbit becomes larger, there is less error between the radius of the orbit measured by convoy range, and r_0. Also as r_0 increased, the collision percentage decreased. There was a noticeable effect on steering energy as r_0 decreased similar to the effect α had on convoy range. When r_0 was 50, steering energy was around 11%. However, once r_0 increased, steering energy dropped to about 2% and stayed constant as r_0 continued to increase. This shows that when the vehicles maintain a small orbit, they must use significantly higher amounts of steering energy. However, as r_0 is increased beyond 50, the vehicles maintain a much wider

orbit which requires minimal steering energy to maintain. Once r_0 is greater than the minimum collision distance, increasing α has a stronger affect on steering energy.

9.3.6. Comparison of 2 and 3 Vehicles

The global minimums for the 2 and 3 vehicle cases were then analyzed to determine if a form of gain scheduling would be appropriate when switching between these situations. For all the 2 vehicle trials, parameter set 1 appeared 6 times, set 6 appeared nine times, seven appeared 3 times, 8 appeared 3 times, and 12 appeared 15 times. Parameter sets 1, 6, and 7 were all top performing parameter sets for the 2 and 3 vehicle cases combined. Parameter set 12 was dominant in the 2 vehicle trials even though it did not appear in the set for all trials. For all of the 2 vehicle trials, α ranged from 0.05-0.35, η ranged from 0.05 to 0.5, and r_0 was always 50. Overall, these gain values were more aggressive than those of the parameter sets for all trials combined. With fewer vehicles than the 3 vehicle case, there is less chance of a collision and less steering energy used so the gain values are able to fluctuate into higher regions. While the α and η values were fairly insensitive, it was again seen that for the best performance, r_0 of 50 was the best choice. For all of these parameter sets, there was zero chance of collision. The trend that as α increases, range to convoy decreases while steering energy increases was also apparent. It was also noticed that when α was constant and η increased, there was an increase in convoy range. This effect was not noticed when all the trials were averaged together, so it may show that η plays a stronger role when only 2 vehicles are used compared to 3 vehicles.

When the 3 vehicle trials were averaged together, parameter set 1 appeared ten times, set 2 appeared twice, set 6 appeared 12 times, and set 7 appeared twelve times. α ranged from 0.05-0.2, η from 0.05-0.35, and r_0 was 50. These values were also identical to the parameter ranges for all sixteen trials averaged together. When compared to the 2 vehicle trials, the 3

vehicle trials had a higher average convoy range and steering percentages and all but one parameter set had zero collision percentage. As more vehicles are added, the stable orbit increases in size and more steering energy is needed to maintain the orbit. This seems contradictory to previous results that showed that as orbit size increases, steering energy decreases. For the 3 vehicle cases, this is true. But when trials 6 and 7 were compared between the 2 and 3 vehicle cases, their convoy range was about the same but for the 3 vehicle case, there was a noticeable increase in steering energy. This may be due to the period of time for the vehicles to get into position. As more vehicles are added to the swarm, more maneuvering is necessary before the vehicles can find the steady state orbit.

When determining if gain scheduling is necessary, parameter set 6 and 7 were compared for the 2 and 3 vehicle case. These sets appeared in the 2 and 3 vehicle cases as well as when all the trials are averaged together. Both perform well regardless of the number of vehicles in the swarm and cost function weighting. While parameter set 12 appeared more for the 2 vehicle case, it did not appear at all in the 3 vehicle case. In the 3 vehicle trials, parameter set 12 was able to keep the vehicles close to the convoy, but at the cost of a noticeable increase in steering energy which prevented it from appearing as a top trial for 3 vehicles. Parameter set 12 had very close performance to both parameter sets 6 and 7. While sets 6 and 7 may not be the absolute best for every cost function weighting permutation, they still performed well. This fact, combined with their performance in the 3 vehicle trials would conclude that choosing either of these parameter sets would be safe for both the 2 vehicle and 3 vehicle case. These sets would also still perform strongly if the swarm size was manipulated during the mission. Due to these results, gain scheduling would not produce any noticeable benefits for the circling scenario.

9.3.7. Comparison of Route 3 and Route 4

The top performing parameter sets were then compared for the two separate routes to see how the swarm responded to changes in direction and varying convoy speeds. Route 3 had parameter set 1 six times, 2 three times, 6 fourteen times, and 7 thirteen times. When route 4 was averaged together, parameter set 1 appeared 8 times, set 2 appeared once, set 6 appeared fourteen times, and set 7 appeared thirteen times. α ranged from 0.05-0.2, η from 0.05-0.35, and r_0 was 50. These were the same trials that were global minimums for all 16 trials averaged together and had the same parameter ranges.

From these results, it is evident that the amount of maneuvering the convoy performs has no strong impact on the performance of the swarm. Both routes had the same parameter sets showing that the swarm performance is insensitive to the convoy route. This is much different than the rectilinear case where the route was a large factor in the ability of the swarm to follow the convoy effectively. If a mission planner was preparing to use UAV swarms for an all foot patrol mission, they would not have to worry about the complexity of the route if the loitering formation was used.

10. Obstacles En Route - Blended Control Investigation

In this scenario, the main objective was to create a more realistic convoy trajectory with varying speeds which represented practical military convoy characteristics. In the real world, military convoys do not always travel at a constant speed. When traveling in and urban environment, especially in a conflict zone, unforeseen events may cause the convoy to slow down or even stop progress. In these situations, the UAVs must still be able to provide accurate and reliable security in the form of sensor coverage. If only the loitering form of the control law

was used, the vehicles would quickly fall behind when the convoy moves at a high speed. However, if only the rectilinear form of the control law was used, the vehicles would scatter and move in a disorganized and inefficient fashion when the convoy speed slows significantly or stops suddenly. To solve this control issue and in order to asses the ability of the control law to operate in a more realistic convoy scenario, a blending of the rectilinear and loitering forms of the control law was investigated. This blending was based on several factors including the speed of the convoy, the distance between the vehicles and the convoy, and the turn bearing rate of the convoy relative to the UAVs.

10.1. Developing Blending Functions

In order to find the best blending method, several blending functions were used. These varying functions determined how quickly or slowly the blending transition between loitering and rectilinear occurred. Based on the speed of the convoy at each time step, a parameter called trans was calculated. The final steering command was then calculated using Equation 4.

$$u = \text{trans} * u\_\text{rect} + (1-\text{trans}) * u\_\text{circ} \qquad [4]$$

When pure rectilinear control is desired, the trans parameter is a value of one. A trans value of 0 is used for pure circular control. These values occur when the convoy speed is below the minimum specified value or above the maximum specified value. In between these minimum and maximum convoy speeds, several different functions were used to calculate the value of trans and apply the blending.

10.1.1. Linear Function

First, a linear interpolation between the minimum and maximum convoy speed values was used. Since the value of trans varies between 0 and 1, the trans value for the linear function was simply the percentage of the current convoy speed relative to the minimum and maximum

specified speeds. This function looks like $trans = \left(\frac{convoy\_speed - \min\_speed}{\max\_speed - \min\_speed} \right)$ and produced the plot of trans values in Figure 24.

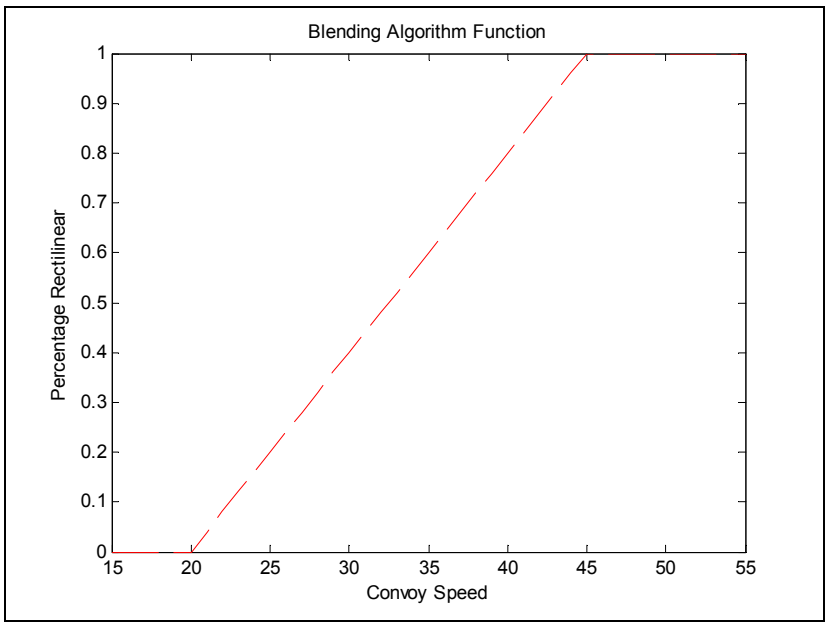


Figure 24: Linear Blending Function

This function results in a linear blend between the minimum and maximum values which were 20 mph and 45 mph for this and subsequent examples unless otherwise stated. It is also clear that varying the minimum and maximum values will change the slope of the line and therefore the behavior of the swarm.

10.1.2. Exponential Function

Next, an exponential function was used which created a function that resulted in a low trans value as the speed increased to a point, then a drastic rise as the speed continued to increase to the maximum user specified speed. The formula used was

$trans = \frac{e^{10*\left(\frac{convoy\_speed - \min\_speed}{\max\_speed - \min\_speed} \right)}}{2.202646579480672^{10000}}$ which resulted in the plot in Figure 25.

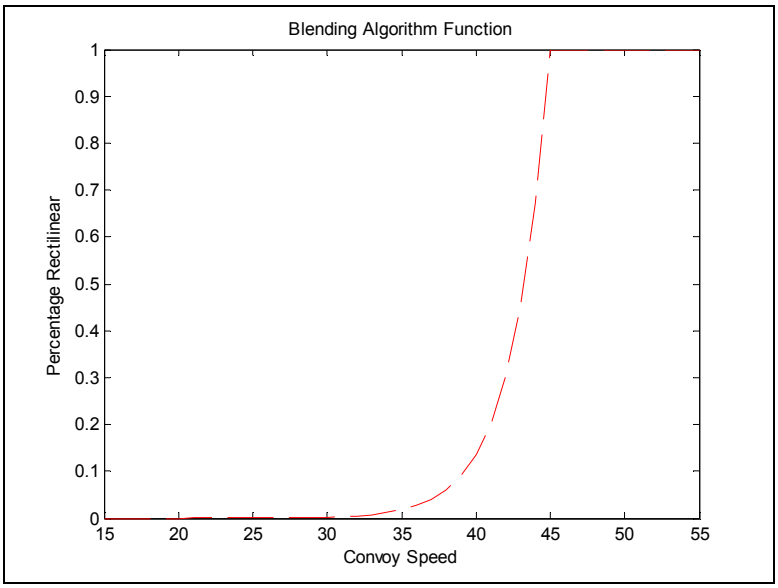


Figure 25: Exponential Blending Function

The specific values used in the formula were chosen on a trial and error basis to enable to function have a minimum value of 0 and a maximum value of 1 for the trans parameter. It is evident that there is little change in the trans value between convoy speed of 20 mph and 35 mph. However, after the convoy speed surpasses 35 mph, the trans parameter increases drastically and maximizes at a value of 1 at a convoy speed of 45 mph.

10.1.3. Logarithmic Function

The next function tested was a logarithmic function that was essentially the inverse of the exponential function. As the convoy speed increased, the trans parameter quickly increased and then settled near 1 as the convoy speed approached the maximum blending speed. This function would show how the behavior of the swarm would react when a greater emphasis is placed on the rectilinear form of the control law. The function used for this was

$$trans = \frac{\log\left[10^{*\left(\frac{convoy\_speed - \min\_speed}{\max\_speed - \min\_speed}\right)}\right] + 2.303}{4.606}$$ and produced the plot in Figure 26.

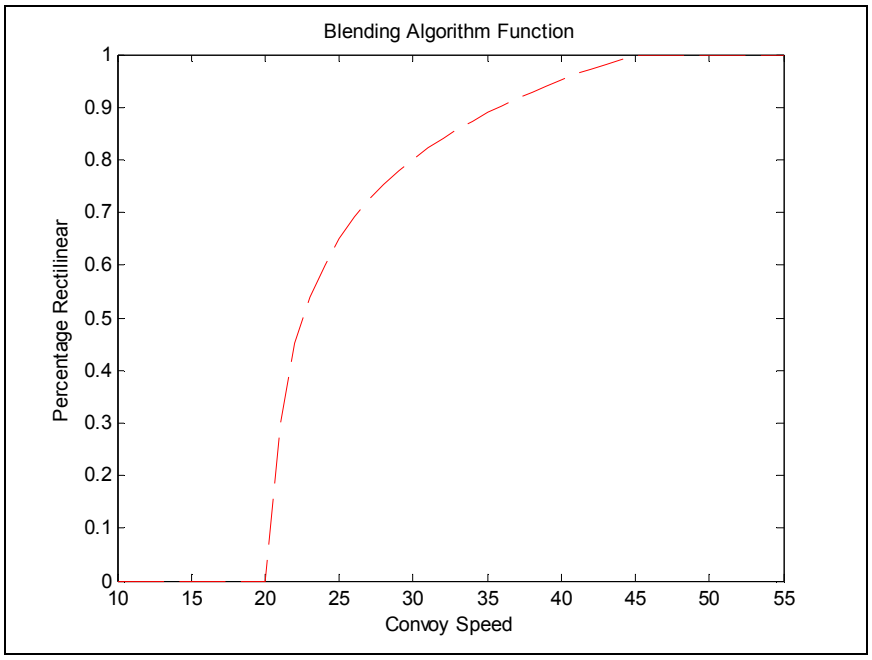


Figure 26: Logarithmic Blending Function

As the convoy speed exceeds the minimum value of 20 mph, the trans parameter quickly increases and eventually settles at a value of 1.

10.1.4. Hyperbolic Tangent Function

Lastly, the hyperbolic tangent was used to effectively combine the effects of the exponential and logarithmic functions. The hyperbolic tangent creates little variation in the trans parameter near the maximum and minimum blending speeds. However, in the middle of the speeds, the trans value quickly increases. The function

$$trans = \frac{1+\tanh\left[5+\left(\frac{convoy\_speed-\min\_speed}{\max\_speed-\min\_speed}\right)-2.5\right]}{2}$$ was used to create the plot shown in Figure 27.

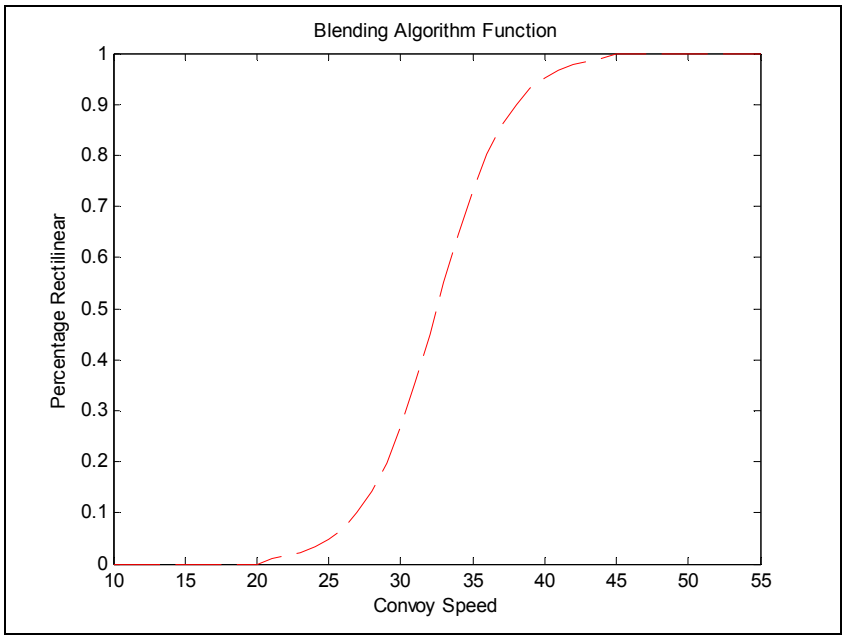


Figure 27: Hyperbolic Tangent Blending Function

This function would show if a symmetric function with a steep initial slope would provide better swarm behavior in contrast to the skewed increases of the exponential and logarithmic functions and the smooth transition of the linear function.

10.2. Speed Blending

This first factor to be analyzed for the basis of blending was the speed of the convoy. This has a very large impact on the effectiveness of the UAV swarm as shown in the previous plot. This blending method assumes that the vehicles all begin in close range with the convoy.

10.3. Speed Limitation Investigation

It would be expected that due to the circling nature of the formation, changes in route direction would have little impact on the vehicles. What would have an impact on the stability of the swarm loitering formation would be the convoy speed. If the convoy is moving too fast, the swarm will not be able to loiter effectively around the convoy. This relationship becomes more important as the size of the orbit decreases. When the orbit is small, the swarm has to respond

quicker to changes in the convoy position. When the orbit is large, the relative position changes of the convoy are small when compared to the size of the orbit. For these reasons, it was necessary to find the limiting speeds of the circular and rectilinear form of the control law

In order to evaluate the effectiveness of the blending schemes and their ability to control the vehicles, a new convoy trajectory was created. This trajectory combined the GPS points of two separate routes; the route from the suburbs to NRL and from NRL to USNA. This longer track provided more time and distance to observe the UAV swarm's behavior. The goal for this trajectory was to have the convoy begin at a slow speed at which point the UAVs could use the pure loitering form of the control law. As the convoy moved along the trajectory, the convoy speed was incrementally increased until the convoy speed reached a point where only pure rectilinear control would allow the swarm to cover the convoy effectively. Thus, a track was made which began with a convoy speed of 10 mph and ended with a convoy speed of 55 mph using Appendix 7. The UAVs were simulated on this track using only pure loitering and then only pure rectilinear control laws. As shown in Figure 28 of the outputs of these two trials, it is evident that at slower speeds, loitering is more effective, and rectilinear becomes more effective as speed increases.

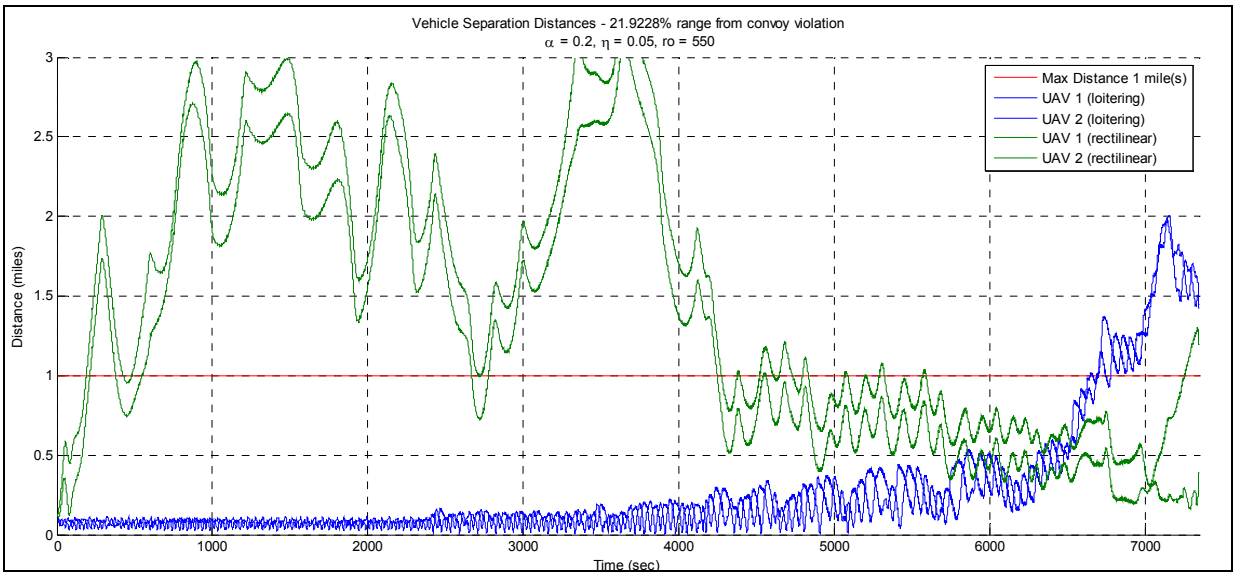


Figure 28: Comparison of Circular and Rectilinear Control

There is a point where both the rectilinear and loitering forms of the control law provide roughly equal quality of performance based on the distance to convoy metric. Using this plot, the minimum and maximum speeds between which the blending would take place were estimated. From this plot, it would be most efficient to transition from circular to blended control once the convoy speed meets 20 mph and for blending to end and transition to rectilinear control once the convoy speed has passed 45 mph. Using these guidelines, a test matrix was created using the values shown in the table in Figure 29. The delta is the value that the speeds were incremented to get a range of values between the respective minimum and maximum speeds.

	Circular Saturation Speed (mph)	Rectilinear Saturation Speed (mph)
Min	10	35
Max	30	55
delta	2.5	2.5

Figure 29: Speed Blend Test Matrix

For all of the simulations, the gain values were chosen from the best performing parameter sets found in the previous scenarios. For rectilinear control, μ was 0.2, α was 0.2, η was 0.05, and r_0 was 550. For loitering control, α and η was 0.2 and r_0 was 50.

10.3.1. Ideal Speed Blending

Data was collected first using the GPS route which had incremental speed increases. In theory, this should allow for the smoothest blending possible and should be the ideal case for each of the blending functions. The data confirmed the assumption that a blending algorithm would perform better than a pure rectilinear or circular algorithm and would be more widely applicable to real world GPS routes. Using pure circular control, the swarm had an average convoy distance of about 22% and this distance increased to 95% when pure rectilinear control was used. However, the majority of the blending algorithms resulted in convoy range percentages averaging around 18%. Clearly, the blending algorithm allowed the swarm to maintain a closer proximity with respect to the convoy as the convoy's speed varied along the route. In this ideal speed varying situation, the convoy's speed changed in equal increments which meant that the blending functions were not perfectly smooth curves.

After collecting the data, a MATLAB script (Appendix 6.3) was used to locate the simulations runs which resulted in the lowest final performance metric as the weights of the cost function were manipulated. The chart (Figure 30) summarizes the important characteristics of the top performing blending functions as the weights of the cost function are manipulated.

Pre-Blended Scenario							
Blending Function	Trial	Times a Global Min	Min Speed (mph)	Max Speed (mph)	Convoy %	Collision %	Steering %
Linear	72	12	27.5	55	14.3627	0	8.902
	24	10	15	47.5	14.6895	0	8.2901
	8	12	10	52.5	15.7229	0	7.5854
	2	2	10	37.5	20.9863	0	6.8227
Exponential	10	1	12.5	35	22.907	0.20862	7.4179
	3	35	10	40	20.6665	0	7.7638
Hyperbolic Tangent	4	3	10	42.5	17.8603	0	7.8989
	5	3	10	45	16.7781	0	8.1091
	34	30	17.5	50	14.1925	0	8.855
Logarithmic	9	3	10	55	26.3548	0	6.4765
	51	1	22.5	47.5	18.1701	0	7.9007
	70	12	27.5	50	15.2661	0	8.4941
	80	20	30	52.5	15.0388	0	8.6858
Averages			16.3461538	46.9230769	17.9227	0.01604	7.938615
Pure Rectilinear					95.208	0	9.7309
Pure Circular					21.9228	1.102	9.814
Overall Top Performer	34		17.5	50	14.1925	0	8.855

Figure 30: Summary of Ideal Speed Blending

When plotted, these functions produce the plot in Figure 31.

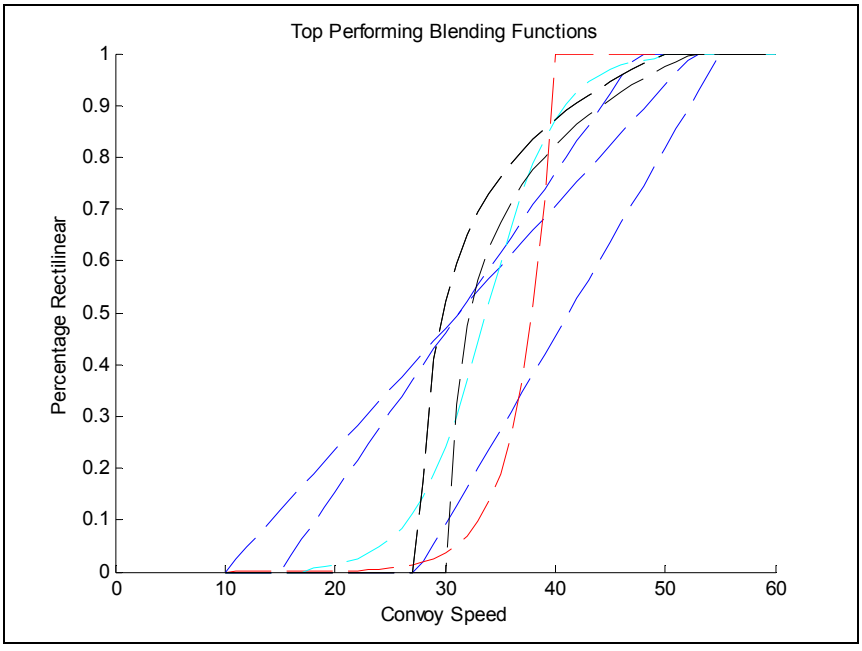


Figure 31: Plot of Top Performing Speed Blending Functions

From these results, it is possible to make several basic conclusions about the best possible method of blending. Most notably is the clear performance advantage that speed blending gives to the behavior of the swarm when convoy speed is non-constant compared to pure rectilinear or circular control. While it appears that pure circular control is sufficient, the UAV swarm begins to quickly fall behind the convoy as the convoy speed increases at the end of the simulation run. Thus, having a blending function that takes this acceleration of the convoy into account and transitions to a more rectilinear based control allows the UAV swarm to keep pace with the convoy as its speed varies.

Of the blending functions, it appears that with the right minimum and maximum speeds set, similar performance appears. Overall, the performance across all range of speed limits did not change drastically and it is not evident that the blending algorithms are extremely sensitive to these bounds. This is also evident when analyzing the plot of the top performing blending functions plotted together. There are no clear trends or distinct overlaps and all seem to fit into a fairly wide band. It may be seen that as the minimum speed limit of the function increases, the initial slope of the blending function increases as well. This suggests that when the minimum speed is set low, the functions allow a gradual transition to rectilinear control. However, when the minimum speed is set high, the blending algorithm must quickly transition from pure circular control to a blend that is more influenced by rectilinear control in order to keep up with the already fast moving convoy. What is clear is that speed blending is an important method to improve the behavior of the swarm and is simple to do computationally.

10.3.2. Practical Speed Blending

For this simulation run, the raw version of the ideal GPS track was used that began in a DC suburb and ended at NRL. This version did not contain any filtering of stop points or speed

modification by manipulating GPS timestamps. This is a more practical and realistic case compared to the ideal situation where the convoy speed steadily increased and the blending could occur smoothly. For this track, the control algorithm would have to adapt to a constantly changing convoy speed and still provide effective convoy security. The same test matrix and blending functions were used for this set of simulations. The data was analyzed by identifying the top performing speed permutations for each blending function as the weights of the cost function were modified. The results are displayed in the table and plot in Figure 32 and 33 respectively.

Suburb-USNA							
Blending Function	Trial	Times a Global Min	Min Speed (mph)	Max Speed (mph)	Convoy %	Collision %	Steering %
Linear	6	3	10	47.5	42.554	0	6.8804
	11	25	12.5	37.5	41.002	0.15438	7.4131
	13	6	12.5	42.5	42.946	0	6.7657
	21	2	15	40	42.177	0	7.0468
Exponential	57	2	25	40	58.157	1.245	6.7781
	58	32	25	42.5	57.411	1.245	6.9283
	62	2	25	52.5	58.353	1.4442	6.6645
Hyperbolic Tangent	2	36	10	37.5	41.611	0	6.9887
Logarithmic	3	3	10	40	43.365	0	7.4421
	6	3	10	47.5	46.332	0	6.8463
	35	30	17.5	52.5	42.486	0.15438	7.4482
Averages			15.681818	43.636363	46.945	0.3857	7.01838
Pure Rectilinear					62.955	0	3.7714
Pure Circular					58.284	1.4143	6.9093
Overall Top Performer	11		12.5	37.5	41.002	0.15438	7.4131

Figure 32: Summary of Suburb to USNA Speed Blending

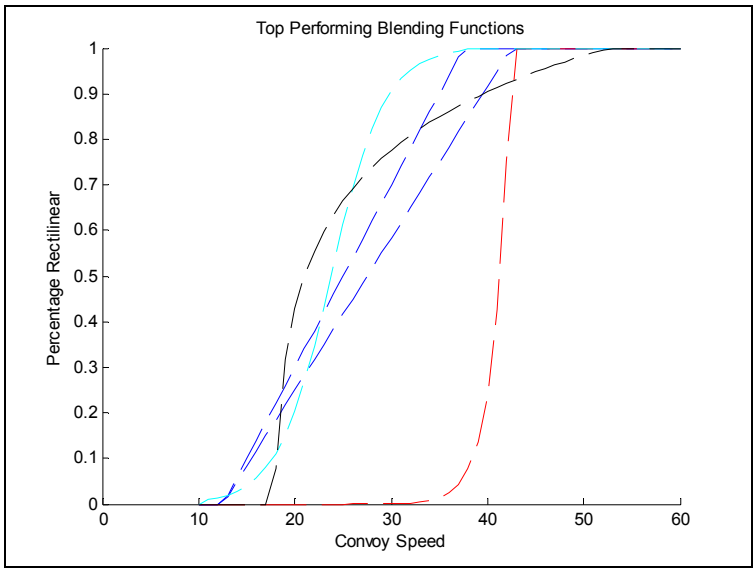


Figure 33: Plot of Top Performing Speed Blending Functions

As is evident from the plot of the top performing blending functions, the practical simulations resulted in a fairly narrow range of top performing blending functions with the exception of the exponential function. The band of top performing blending sets appears slightly narrower in this route when compared with the pre-blended route used before. Since this simulated route is a purely raw track, these top performing blending trials may be better suited for raw GPS routes. Also, the blending for this raw track began on average at a lower speed and the blending ended on average at a lower speed compared to the ideal case. The blending needed to transition to rectilinear earlier and reach pure rectilinear control faster in order to keep up with the convoy accurately. Ultimately, the performance across the various blending functions was very close which supports the previous conclusion that swarm performance is not sensitive to the specific function and the speed limitations of the blend.

Performance wise, the practical simulation had significantly higher performance percentages compared to the ideal situation which is to be expected. The methodical blending used in the first simulation allows for a smooth transition between circular and rectilinear control

whereas the raw track causes much more abrupt transitions between circular and rectilinear. Also, in the pre-blended track, the convoy moves at a slow speed compared to the UAV max speed for the majority of the simulation. This is not the case in the raw track, since the convoy's speed varies constantly and erratically. Lastly, it is of note that with this more realistic GPS route, the speed blend still outperformed the pure rectilinear and pure circular control. This is more evidence to suggest that blending is an effective control method that should be pursued. It is believed that as long as some form of blending is used with appropriate speed limitations, the result will be improved swarm performance.

10.3.3. Analysis of Additional Routes

In order to gain a broader perspective on the performance of the speed blending method, two additional real world GPS tracks were used to analyze the performance of the speed blending method. The first track comprised of a route from NRL to a house in Northern DC on Van Ness Street. The second track is a route from the house in Northern DC back to USNA. The results for speed blending in the route from NRL to Van Ness are depicted in Figure 34 and Figure 35.

NRL to VN Blending Function	Trial	Times a Global Min	Min Speed (mph)	Max Speed (mph)	Convoy %	Collision %	Steering %
Linear	1	4	10	35	44.946	0.28937	6.0557
	4	17	10	42.5	44.677	0.08064	6.4549
	13	3	12.5	42.5	46.777	0.14706	5.328
	14	12	12.5	45	45.923	0	5.5749
Exponential	1	1	10	35	53.451	1.9165	5.8103
	7	35	10	50	51.912	1.518	6.1378
Hyperbolic Tangent	1	34	10	35	43.084	0	6.393
	2	1	10	37.5	46.202	0.0948	5.878
	5	1	10	45	50.830	0.0948	5.295
Logarithmic	18	2	12.5	55	45.892	0	5.4253
	19	34	15	35	39.770	0	6.3923
Averages			11.136363	41.5909090			
Pure Rectilinear					63.474	0	3.8508
Pure Circular					55.798	5.2989	6.4702
Overall Top Performer	19		15	35	39.770	0	6.3923

Figure 34: Summary of NRL to Van Ness Speed Blending

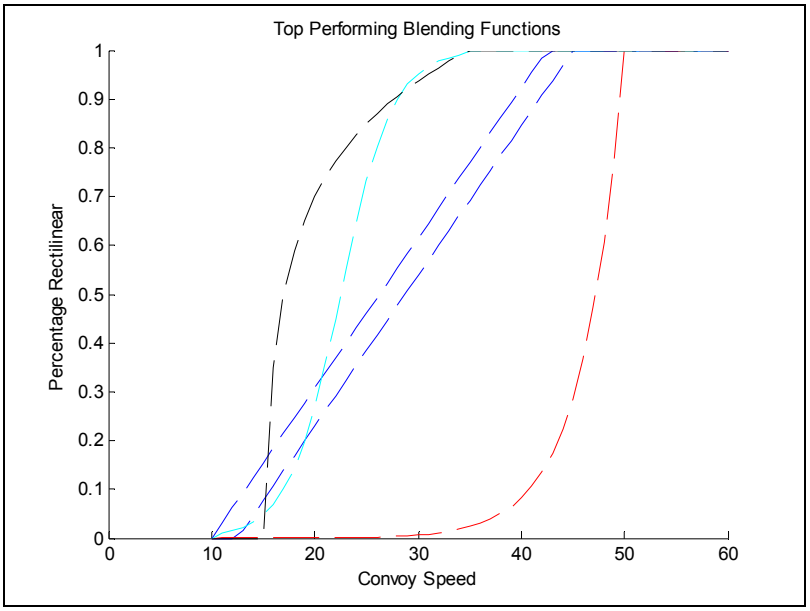


Figure 35: Plot of Top Performing Speed Blending Functions

Again, the performance numbers for the blending functions are all relatively similar and all out perform the pure rectilinear and circular control methods. For this route, the linear function had relatively good performance, but one logarithmic function ended up being the best

overall performer for this route. The results from the Van Ness Street to USNA track are shown in Figure 36 and Figure 37.

VN to USNA Blending Function	Trial	Times a Global Min	Min Speed (mph)	Max Speed (mph)	Convoy %	Collision %	Steering %
Linear	10	4	12.5	35	23.1941	0	9.2997
	19	7	15	35	23.0779	0	9.3311
	20	24	15	37.5	22.8733	0	9.4391
	44	1	20	52.5	24.9572	0.09498	9.0418
Exponential	6	4	10	47.5	23.4565	0.86673	10.0246
	10	26	12.5	35	24.0986	0	9.3807
	15	6	12.5	47.5	23.6872	0.27011	9.7838
Hyperbolic Tangent	1	12	10	35	23.3103	0	9.2734
	3	4	10	40	22.826	0	9.4143
	10	18	12.5	35	22.826	0	9.4143
Logarithmic	33	8	17.5	47.5	22.6126	0	9.346
	35	27	17.5	52.5	22.2784	0	9.4917
	45	1	20	55	23.8679	0.08607	9.153
Averages			14.2307692	42.6923076	23.3127	0.10137	9.414885
Pure Rectilinear					60.1059	0	4.8993
Pure Circular					24.1143	0.48679	9.8864
Overall Top Performer	35		17.5	52.5	22.2784	0	9.4917

Figure 36: Summary of Van Ness to USNA Speed Blending

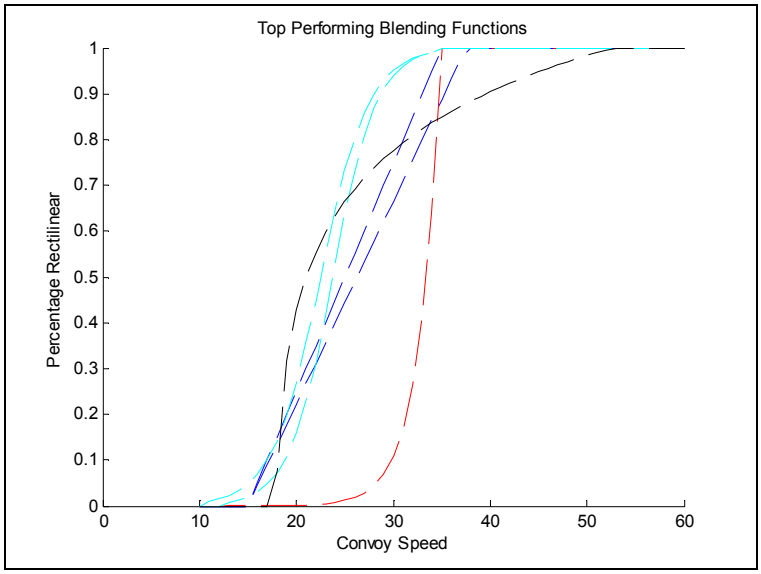


Figure 37: Plot of Top Performing Speed Blending Functions

These results repeated the trends seen throughout the speed blend data. All of the blends have very close performance metrics, showing that the type of blending function is generally irrelevant in determining the resulting behavior of the swarm. In the Van Ness to USNA track, the pure circular control is very close to the blending performance. This is due to the slow convoy speed for this route which results in the blending function relying mostly on pure circular control.

For these two additional routes, the top performer ended up being a logarithmic function. However, since all of the performance metrics are so similar, there is no strong evidence to conclude that a logarithmic blend is better than the other functions. Thus, there was no specific blending function that performed better over the range of routes throughout the speed blending simulations. Still, the simulation results suggest that an exponential blending function is a poor blending method. Even the top performing exponential blends had noticeably poor performance compared to the other blending methods. Examining the plots of the blending functions shows that the exponential function is typically displaced from the other functions. While the linear, hyperbolic tangent and logarithmic functions are not in a very precise band, they are all in the same relative position when plotted. The exponential function may blend at the same relatively low convoy speeds like the other functions, but it takes longer for the impact of increased convoy speed to have a noticeable effect on the transition to rectilinear control. When the slope does increase, the convoy speed is typically at a higher speed relative to the other blends which means the UAVs are falling behind the convoy as they transition to rectilinear. This results in increased convoy range percentage and poorer performance. For GPS routes where the convoy moves at quicker speeds such as in the Suburb to USNA track and the NRL to Van Ness Street track, this

effect is more pronounced and can be seen when comparing the convoy range percentages of the exponential blends and the other blending functions.

Also, looking at the average minimum and maximum speeds that determine the blending limits, the minimum speeds tend to be towards the lower end of the speed range while the maximum speeds are more towards the middle of the speed range. There also seems to be a slight correlation between the convoy range percentage and the average transitions speeds. The higher the average distance to convoy percentage, the lower the average minimum and maximum speeds of the blending limits are. The routes that have lower convoy range percentages tend to have convoys that move at generally lower speeds. By looking at the average convoy range percentage for the pure circular and the overall top performer case, one can tell how slow the convoy was moving. The closer these percentages are, the more the speed blend was using circular control, and thus the slower the convoy was moving. The NRL to Van Ness track begins with a fast highway section that leaves the UAVs behind. Eventually, the convoy hits traffic in the city, slows down, and the UAVs catch up. It seems logical then that as the convoy moves faster, the blend would want to transition to rectilinear control faster, hence the blends with lower speeds cut offs appear as top performers. These lower transition speeds allow the UAVs to reach maximum speed with the rectilinear control faster, and thus enable the UAVs to reach the convoy in a faster manner.

10.4. Distance Blending

Speed blending proved to be an effective technique for improving the behavior of the swarm as the vehicles followed a convoy of constantly fluctuating speed. However, the blending function had one assumption which is not always true on the battlefield. For the speed blending, the UAVs all started relatively close to the convoy. In the real world, it may not be possible to

launch the UAVs near the convoy. To support a wider variety of missions, a form of distance blending was also implemented. For example, if the UAVs began several kilometers away and were tasked with following a slow moving convoy using speed blending, they would use predominantly circular control which would prevent the vehicles from reaching the convoy in a timely manner. What would be desired in this case is for the vehicles to use rectilinear control to quickly close the distance to the convoy then switch to speed blending once in the immediate vicinity of the convoy.

With this goal in mind, distance blending was implemented in to the control of the UAV swarms. When the UAVs were outside a certain user specified distance, pure rectilinear control was used. As the UAVs get closer to the convoy, the distance blending formula blends rectilinear control with speed blending control. Once inside a certain distance, only speed blending control is used. For this method, the values shown in Figure 38 were used.

	Minimum Blending Distance (miles)	Maximum Blending Distance (miles)
Min	0.1	0.8
Max	0.5	1.2
delta	0.5	0.5

Figure 38: Distance Blending Test Matrix

This blended control method produced the chart in Figure 39 depicting the top performing blends for each blending function, as well as the results for pure rectilinear, pure circular, and pure speed based control for comparison purposes.

Suburb-USNA Blending Function	Trial	Times a Global Min	Max Dist (miles)	Min Dist (miles)	Convoy %	Collision %	Steering %
Linear	28	14	1.05	0.1	34.893	0	7.286
	59	12	0.9	0.3	34.2899	0	7.8222
	74	10	0.8	0.15	34.407	0	7.6214
Exponential	6	11	1.2	0.35	34.9779	0	7.7169
	28	17	1.1	0.5	35.0984	0	7.5635
	43	4	1	0.4	34.8936	0	8.1246
	47	4	0.95	0.15	35.6369	0	7.4367
Hyperbolic Tangent	39	2	1	0.2	35.1561	0	7.218
	74	34	0.8	0.1	33.8266	0	7.4091
Logarithmic	1	4	1.2	0.1	48.9129	0	3.7878
	68	12	0.85	0.3	34.2013	0	7.4872
	70	20	0.85	0.4	34.2696	0	7.3663
Averages			0.975	0.25416666	35.8802	0	7.236642
Pure Rectilinear					62.9555	0	3.7714
Pure Circular					58.285	1.4143	6.9233
Pure Speed Blend					41.0029	0.15438	7.4131
Top Overall Performer	74	34			33.8266	0	7.4091

Figure 39: Summary of Suburb to USNA Distance Blending

From the chart in Figure 39, it is clear that adding this distance condition to the blend created swarm behavior that is much more desirable. The vehicles stay significantly closer to the convoy with this addition. All of the top performing distance blending trials, vice one logarithmic trial, have lower convoy range percentages than speed control and pure circular and rectilinear control. Again, all of the performance metrics are very similar leading to the conclusion that the type of blending function is fairly irrelevant for distance blending as well.

A comparison of a simulation run with distance blending and then a plot of the same run without distance blending is depicted in Figures 40 and 41. Here it is evident that the distance blending brings the UAVs closer to the convoy faster when the UAVs are placed at a distance of over a mile away from the convoy. In these simulations, the swarm using distance blending gets under the red line, meaning that the vehicles are now in sensor range of the convoy, in about 300

seconds. In the plot without the distance blending, the swarm takes almost twice as long, about 600 seconds, to get to within effective range of the convoy. Also from this plot, it is evident that in cases where the convoy vehicle speeds up and the swarm falls behind, distance blending forces the swarm to rely more on rectilinear control faster than pure speed blending. This can be seen at the end of these simulation plots where the swarm without distance blending begins to fall out of convoy range compared to the swarm with distance blending which remains in convoy range.

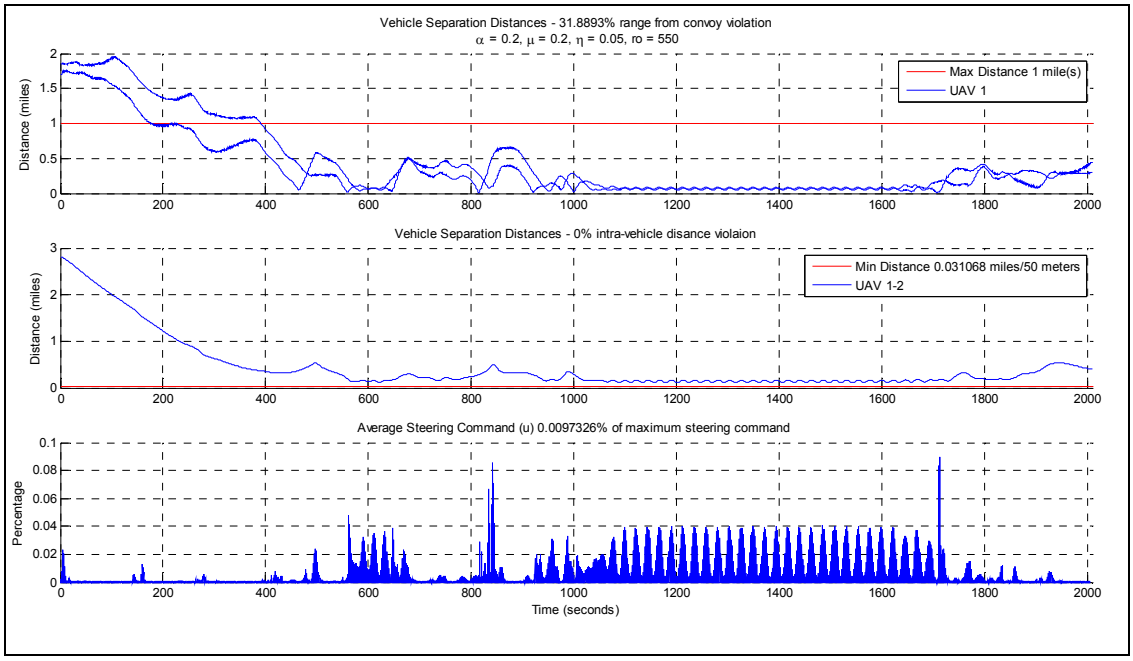


Figure 40: Simulation using Distance Blending

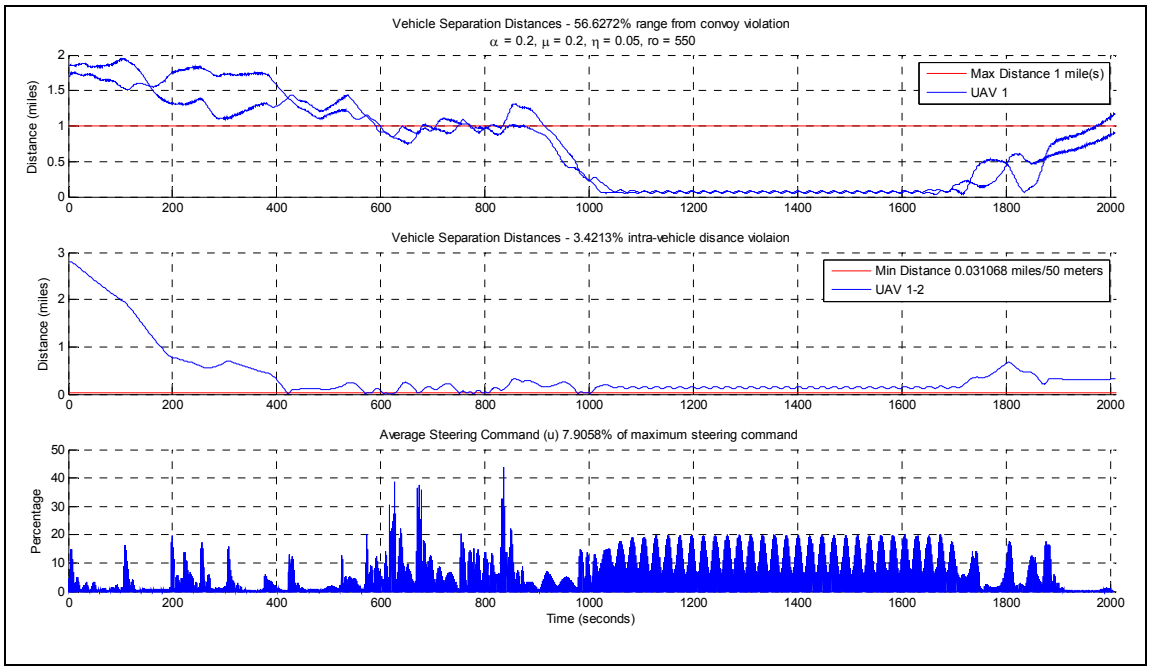


Figure 41: Simulation using only Speed Blending

10.4.1. Analysis of Additional Routes

A similar method of analysis was applied to the data for the use of distance blending on the additional GPS routes. The final data tables for the Van Ness to USNA route and the NRL to Van Ness route are shown in Figure 42 and Figure 43.

VN to USNA							
Blending Function	Trial	Times a Global Min	Max Dist (miles)	Min Dist (miles)	Convoy %	Collision %	Steering %
Linear	3	1	1.2	0.2	28.459	0.09795	8.9426
	43	29	1	0.4	27.295	0.01238	9.2865
	63	1	0.9	0.5	27.580	0	9.5668
	64	1	0.85	0.1	29.189	0	8.8015
	65	4	0.85	0.15	30.257	0.07123	8.5114
Exponential	30	2	1.05	0.2	29.867	0.12763	8.8786
	61	5	0.9	0.4	29.197	0.02077	9.0383
	64	9	0.85	0.1	28.972	0.07123	9.2611
	66	1	0.85	0.2	28.280	0	9.4615
	78	19	0.8	0.35	28.159	0	9.5657
Hyperbolic Tangent	64	29	0.85	0.1	28.36	0	8.8357
	65	7	0.85	0.15	27.963	0.13951	9.5784
Logarithmic	10	3	1.15	0.1	49.415	0.07717	4.3425
	14	5	1.15	0.3	28.702	0	8.8901
	17	15	1.15	0.45	27.952	0.18106	9.2773
	39	3	1	0.2	34.684	0	7.1241
	72	10	0.85	0.5	28.214	0.10686	9.055
Averages			0.9558823	0.2588235	30.118	0.0567	8.7304
Pure Rectilinear					60.105	0	4.8993
Pure Circular					24.114	0.48679	9.8864
Pure Speed Blend					22.278	0	9.4917
Top Overall Performer	43	29			27.295	0.07123	9.2865

Figure 42: Summary of Van Ness to USNA Distance Blending

NRL to VN Blending Function	Trial	Times a Global Min	Max Dist (miles)	Min Dist (miles)	Convoy %	Collision %	Steering %
Linear	10	1	1.15	0.1	46.232	0	6.3648
	28	5	1.05	0.1	46.999	0	6.0781
	49	27	0.95	0.25	44.0568	0	7.2996
	65	3	0.85	0.15	48.655	0	5.7482
Exponential	48	2	0.95	0.2	48.0894	0	6.8069
	51	9	0.95	0.35	47.7143	0	7.1933
	54	19	0.95	0.5	47.8355	0	6.906
	60	3	0.9	0.35	49.1802	0	6.4894
	67	3	0.85	0.25	51.3239	0	6.0872
Hyperbolic Tangent	1	13	1.2	0.1	46.3949	0	7.3876
	28	4	1.05	0.1	47.4667	0.10436	6.5976
	31	15	1.05	0.25	46.6485	0	6.9942
	35	4	1.05	0.45	50.1234	0	6.12
Logarithmic	20	5	1.1	0.15	54.1215	0	4.3697
	22	30	1.1	0.25	44.6926	0	6.9292
	40	1	1	0.25	48.2714	0	5.8656
Averages			1.009375	0.2375	47.9878	0.0065	6.4523
Pure Rectilinear					63.4749	0	3.8508
Pure Circular					55.7989	5.2989	6.4702
Pure Speed Blend					39.7704	0	6.3923
Top Overall Performer	49	27			44.0568	0	7.2996

Figure 43: Summary of NRL to Van Ness Distance Blending

In these blends, the exponential function is still displaced from the other top performing blending functions as seen in Figure 44 of the top performing functions for the suburb to NRL route.

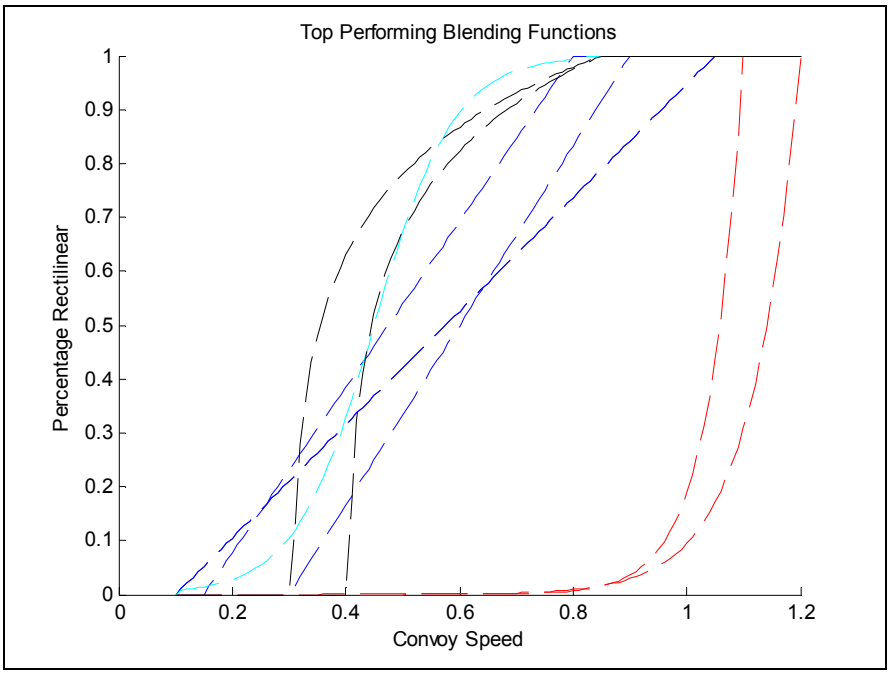


Figure 44: Plot of Top Performing Distance Blending Functions

However, unlike in speed blending, this displacement does not appear to have such an adverse affect on the performance of the swarm. One reason for this is the speed of change of the variable being measured. In speed blending, the blend is based off the convoy's speed which can change from the minimum blending speed of 10 mph to the maximum of 55 mph in a matter of seconds. However, in distance blending, the vehicle dynamics prevent the UAVs from closing the distance to the convoy at such a comparatively quick rate. Thus, in speed blending, the delay in transition to rectilinear control caused by the exponential function is intensified. In distance blending, the convoy to UAV distance variable can not change as fast as the convoy speed, and the detrimental effects of the exponential function are not as apparent. Lastly, similar to the results of the speed blending, the specific function does not seem to have a large impact on the final behavior of the swarm, as evident by the relatively similar performance numbers for the top performing distance blending trials.

In distance blending, there also appeared to be a correlation between convoy speed, as measured by the average convoy range percentage, and the distance minimum and maximums that appeared in the top performing blending trials. As average convoy range percentage increased, the minimum distance value decreased and the maximum distance value increased. When the convoy was moving at a faster speed, the blending functions which started blending sooner, and therefore had larger maximum distance values, performed better. Also, the minimum distance was made smaller, which meant that the blend favored rectilinear control to get the UAVs closer to the convoy before transitioning to full speed control. A faster moving convoy resulted in a wider range between the minimum and maximum distances to allow more rectilinear control in the final steering command which resulted in the UAVs being able to maintain a closer distance to the convoy. At slower speeds, the opposite happened. The transition to speed blending happened when the vehicles were closer to the convoy and transitioned to speed control faster as well. The range was smaller which allowed the UAVs to transition to speed control faster and at a farther distance since the convoy was moving at a slower speed and was unlikely to take off suddenly. When the convoy is moving slow, speed control is a more accurate and effective form of blending as the UAVs get closer to the convoy.

A comparison of the results for the top performing distance and speed blends also revealed some important information about the implementation of distance blending. In order to get the most effective distance blend possible, the parameters for the best speed blending on each track were utilized so that the distance blend would transition from rectilinear to top performing speed blend control. As evident in the tables in Figure 42 and Figure 43, for two out of the three routes, the speed blend outperformed the distance blend. There are several reasons for this. First, in these scenarios, the UAVs were not placed at an extreme distance away from the convoy. This

limited the advantages that distance blending would have in such a situation. Also, if the convoy begins the route moving near the maximum blending speed, the speed control will result in pure rectilinear control and have the same effect as the distance controller outside of the maximum distance range. If the vehicles are placed at a significant range and the convoy is moving at a moderately slow pace, the effect of the distance blending is clearly evident. In the plots in Figures 45 and 46, the UAVs were placed about 6 miles away from the convoy initially, and the convoy moved at a slow pace throughout the simulation. Comparing the two plots, it is evident that the distance component brings the UAVs very close to the convoy before transitioning to speed control. When pure speed control is used, the only variable is the convoy speed and since the convoy is moving fairly slow, circular control is mostly used. This results in noticeably poor performance.

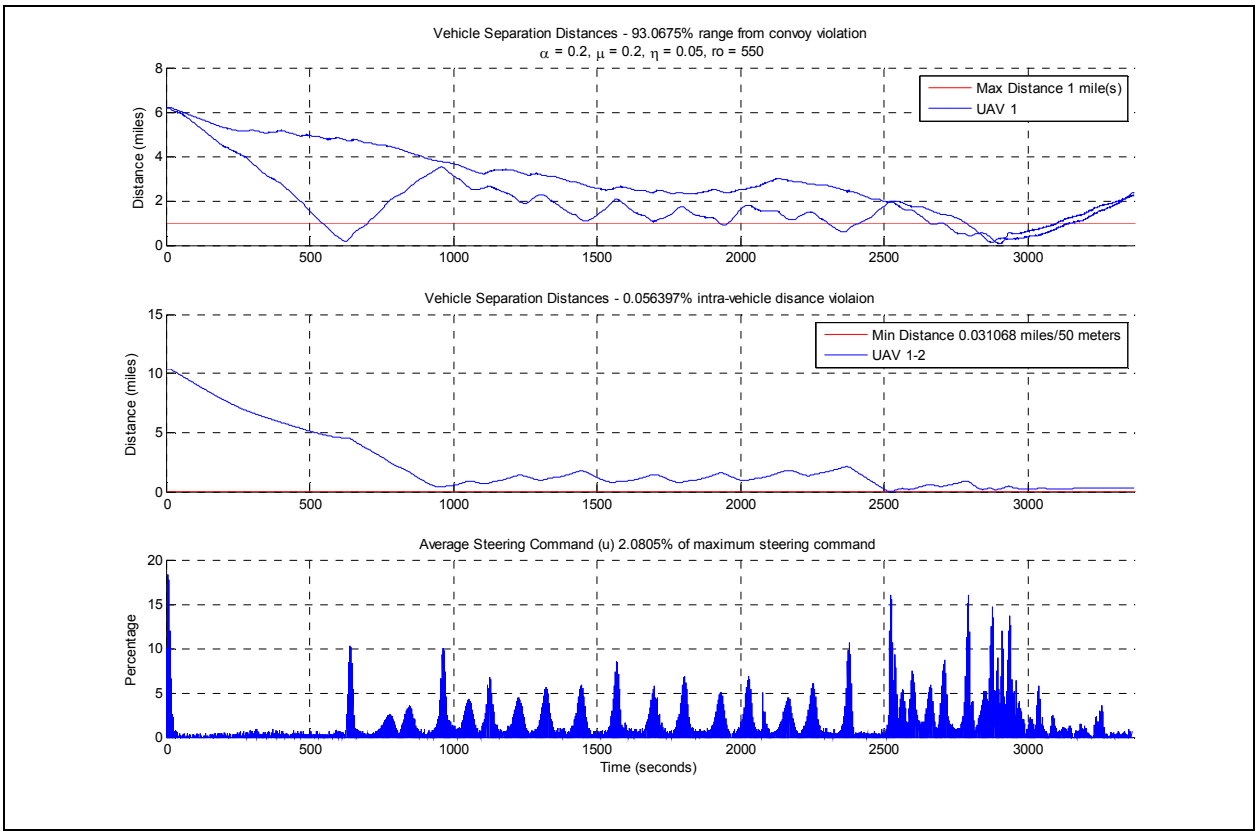


Figure 45: Simulation using Speed Blending

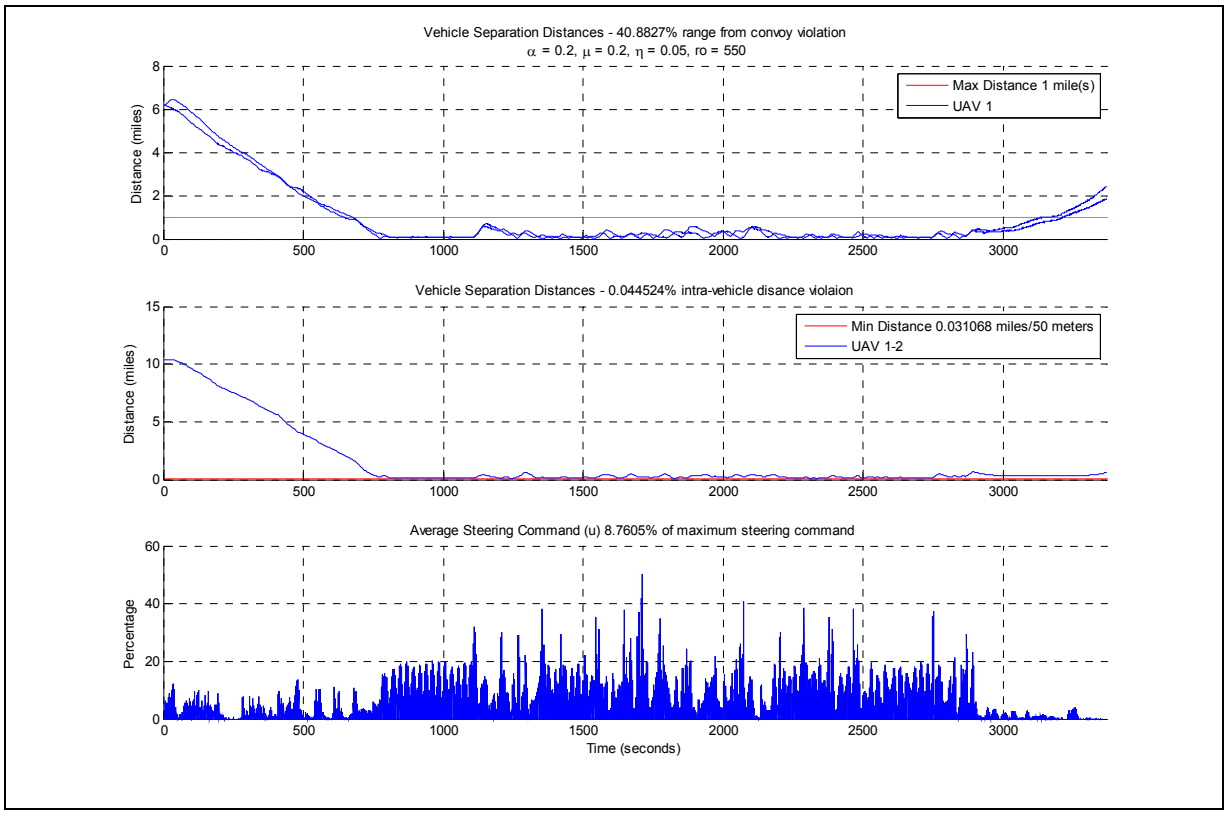


Figure 46: Simulation using Distance Blending

If the vehicles are launched fairly close to the convoy, the advantages of the distance blend are negligible. Part of the reason is that when the vehicles using pure speed control begin to move towards the convoy, they start to converge into a large circular orbit. This, in effect, brings them to the convoy in a fairly straight manner similar to rectilinear control, and allows them to establish a stable circular orbit as they approach the UAV which benefits the performance if the convoy is moving slow and circular control is utilized. However, one way to improve the performance of the distance blending method is to increase the minimum and maximum distances of the blending function. This was done on the route from NRL to Van Ness Street.

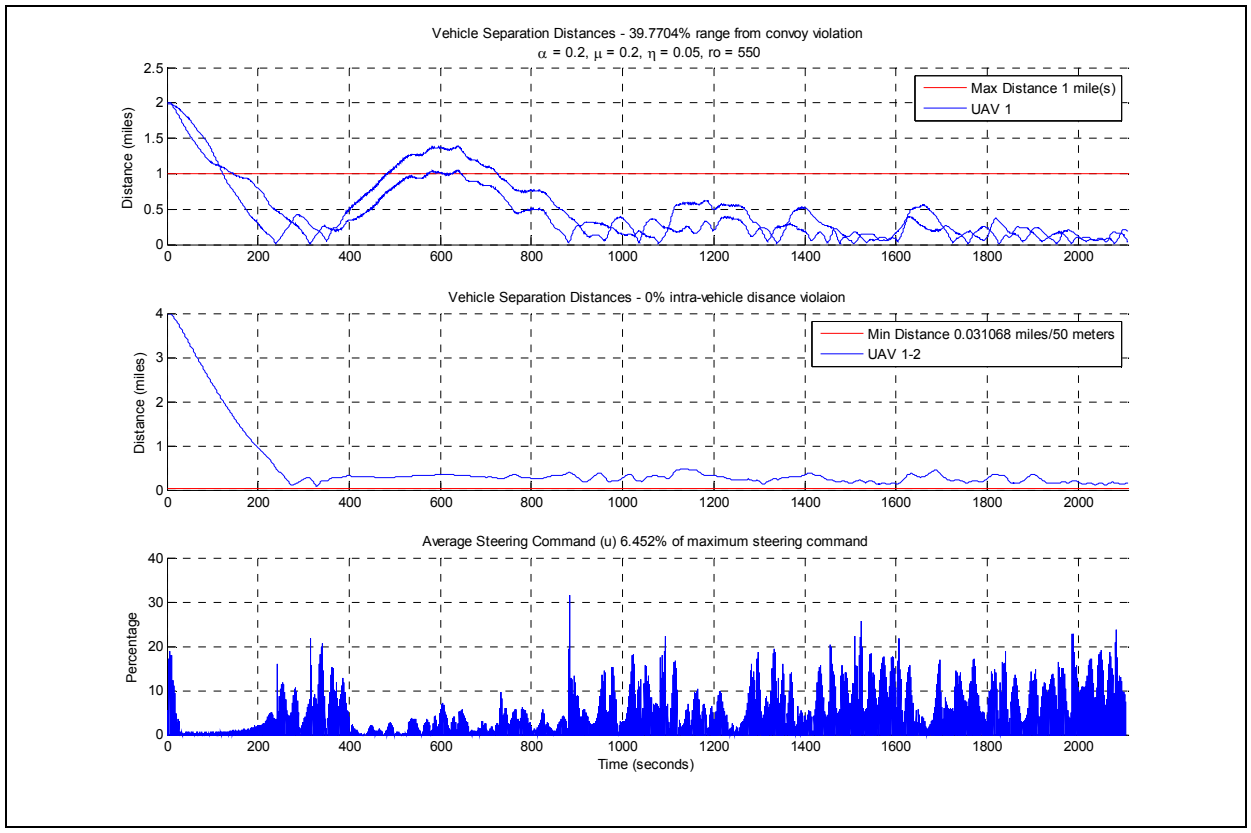


Figure 47: Simulation using Speed Blending

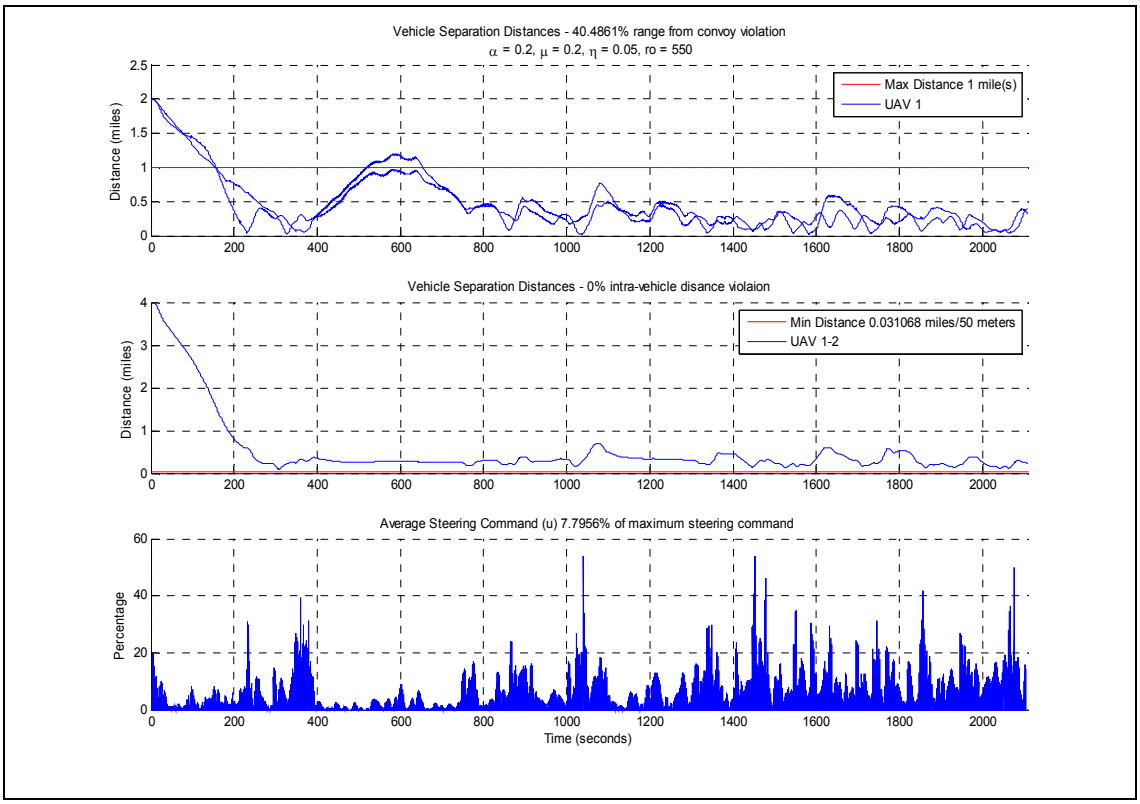


Figure 48: Simulation using Distance Blending

By increasing the blending distances, in effect putting a larger influence on speed blending, performance was increased from the previous top performing distance blending function. Previously, the top performing blending function had a convoy range percentage of 44.0568%, collision percentage of 0%, and a steering energy of 7.2996%. The performance was improved to 40.4861% convoy range, 0% collision, and 7.7956% of maximum steering energy. While further refinement and testing would need to be done to identify the appropriate limits for the new top performing distance values, it is evident that when the UAVs are placed near the convoy, speed or distance blending result in similar performance and the distance blending can be refined for improved behavior if necessary.

10.5. Bearing Rate Blending

The intent behind this form of bended control was to create a method that was effective, but also more robust and requiring less technology to implement. With an electronic warfare defensive mind frame, it is evident that having the military convoy broadcast its GPS position constantly is not operationally secure. The GPS signal can be jammed or degraded by enemy forces, reducing the effectiveness of the swarm. Worse still, the enemy could intercept the GPS signal and have an accurate idea of the position of our troops in a potentially hostile area. Alternate methods of distance calculation can be used between the UAVs and the convoy to determine the straight line distance between the vehicles. This could include, but is not limited to, using an electromagnetic signal and measuring the transfer time between the convoy and the UAV to determine the distance between the vehicles. This would provide the distance measurement without transmitting exact GPS coordinates.

10.5.1. Bearing Rate Calculation

In an attempt to remedy the electronic attack hazard, the method of calculating the relative bearing rate of the convoy to each UAV was developed to blend the circular and rectilinear control laws. This form of blending requires the UAVs to be outfitted with some sort of visual device such as a camera or IR sensor. These are usually available on modern UAVs and do not require any additional hardware to implement this method. The convoy only needs to wear some sort of IR flasher or other visual identification mark for the UAVs to recognize. Most convoys already carry this sort of equipment to prevent blue on blue targeting, so there is no additional hardware needed on that end either. Using these devices, the UAVs are able to locate the relative position of the convoy and determine the rate at which the convoy is moving. This, in effect, combines the distance and speed variables used in previous methods into one simple

calculation. To understand how this works, imagine a ship on the ocean coming into harbor. If one was on the bridge, a lighthouse far in the distance may be seen. This position doesn't change when the ship is far away. However, as the ship maneuvers closer and begins to pass the lighthouse to either side, the relative position of the lighthouse moves and it does so at an increasing rate. In our scenario, the only difference is that the lighthouse is a military convoy which is not stationary, but capable of moving. Still, the concept is the same and the calculation is shown in Equation 5.

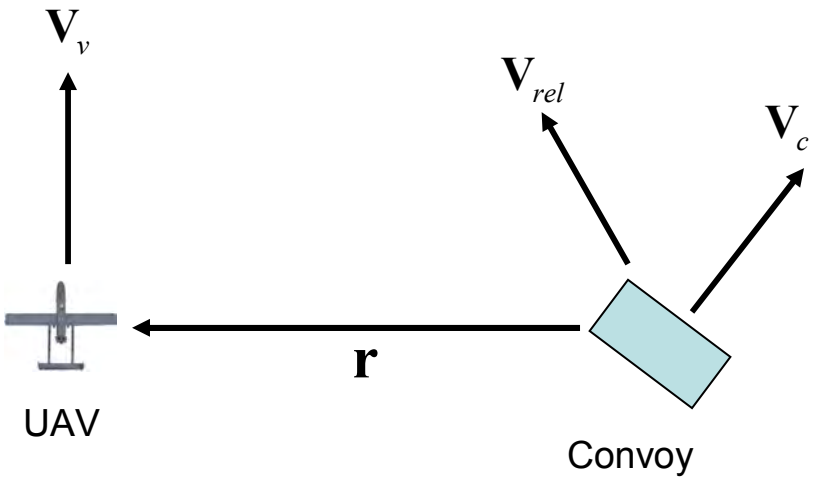


Figure 49: Bearing Rate Blending Model

$$\mathbf{V}_{rel} = \mathbf{V}_v - \mathbf{V}_c$$

$$\dot{\theta}_{rel} = \frac{\mathbf{r}^{\perp} \cdot \mathbf{V}_{rel}}{|\mathbf{r}|^2} \qquad [5]$$

This formula was applied to the Suburb to NRL route and the bearing rate for the vehicles was recorded. This data is represented in Figure 50 showing the bearing rate for a single vehicle in a two vehicle swarm.

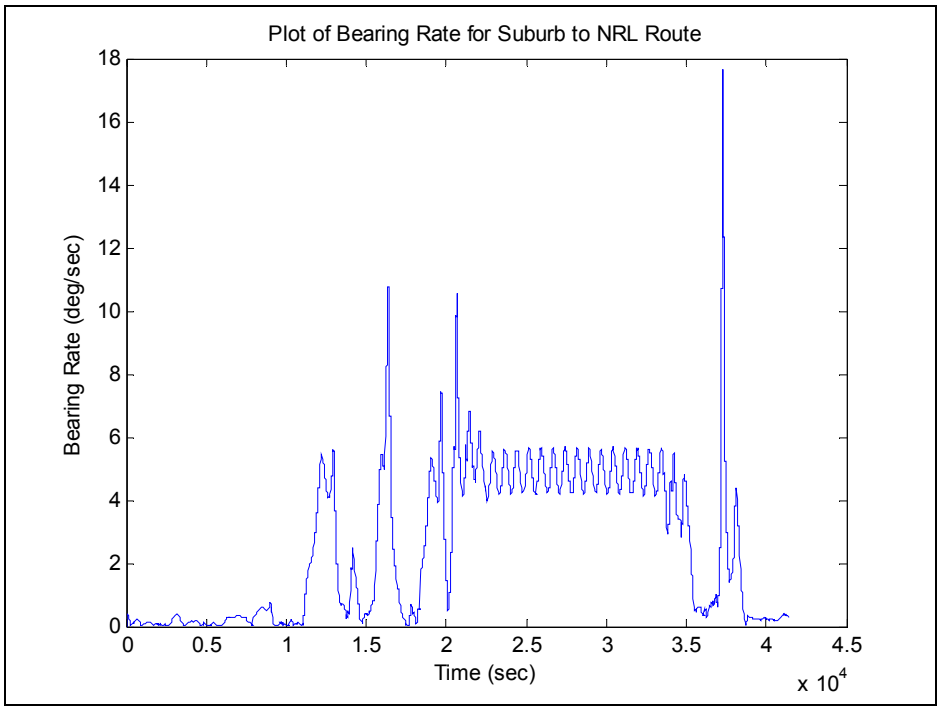


Figure 50: Plot of Bearing Rates

By comparing this plot, it was possible to determine some general bearing rate numbers for loitering and rectilinear control. In the beginning of the plot, the bearing rate is low so rectilinear control would want to be used for bearing rates below about 3 degrees per second. Between 2 and $3.5 \cdot 10^4$ seconds, the bearing rate oscillates which corresponds to the UAVs loitering around the convoy. Therefore, circular control should occur between 4 and 6 degrees per second of bearing rate. With these general guidelines, a test matrix was made (Figure 51).

	Minimum Blending Bearing rate (deg/sec)	Maximum Blending Bearing Rate (deg/sec)
Min	1	3.25
Max	3	5.25
delta	0.25	0.25

Figure 51: Bearing Rate Blending Test Matrix

10.5.2. Data Analysis for Additional Routes

The bearing rate method was used on the same three routes as the previous blending methods. The results for the top performing bearing rate blending functions are summarized in the table in Figure 52 as well as the performance of speed control, distance control, pure circular control and pure rectilinear control for comparison purposes.

Suburb-USNA Blending Function	Trial	Times a Global Min	Max Bearing Rate (deg/sec)	Min Bearing Rate (deg/sec)	Convoy %	Collision %	Steering %
Linear	10	3	3.5	1	36.9241	0	5.8003
	12	33	3.5	1.5	36.6335	0.13944	5.6101
Exponential	9	36	3.25	3	44.6305	0	5.2335
Hyperbolic Tangent	9	36	3.25	3	44.6385	0	5.2415
Logarithmic	6	36	3.25	2.25	31.2597	0	2.3004
Averages			3.35	2.15	38.81726	0.027888	4.83716
Pure Rectilinear					62.9555	0	3.7714
Pure Circular					58.285	1.4143	6.9233
Pure Speed Blend					41.0029	0.15438	7.4131
Pure Distance Blend					33.8266	0	7.4091
Top Overall Performer	6	36	3.25	2.25	31.2597	0	2.3004

Figure 52: Summary of Suburb to USNA Bearing Rate Blending

It is evident from these results that the bearing rate blending function used had a much larger impact on the resulting swarm performance. Also, for the suburb to USNA track, there was much less variability between the top performing blends for each individual function. Only the linear function had two different trials appear as top performers and even then, trial 10 only appeared three times. Overall, the bearing blend outperformed the pure rectilinear and circular control methods significantly and managed to produce better performance than the speed or distance blending on the same route for the bearing rate blend using a logarithmic function.

The results of the bearing rate blending simulations on the NRL to Van Ness Street route are summarized in the table in Figure 53.

NRL to VN Blending Function	Trial	Times a Global Min	Max Bearing Rate (deg/sec)	Min Bearing Rate (deg/sec)	Convoy %	Collision %	Steering %
Linear	10	6	3.5	1	47.4418	0.08538	4.9672
	31	6	4	1.75	47.4987	0	4.8906
	56	24	4.75	1.25	47.7825	0	4.3766
Exponential	2	6	3.25	1.25	50.0774	0	4.6643
	6	5	3.25	2.25	51.1619	0	4.3647
	12	25	3.5	1.5	49.5532	0.20398	4.7159
Hyperbolic Tangent	5	2	3.25	2	51.0638	0.16129	4.2498
	16	15	3.5	2.5	49.2736	0	4.8567
	29	18	4	1.25	49.4683	0	4.6054
	39	1	4.25	1.5	51.1784	0	4.2904
Logarithmic	1	2	3.25	1	45.1924	0.8112	5.5458
	2	2	3.25	1.25	45.0795	1.1717	5.6976
	10	12	3.5	1	45.4565	0	5.3983
	26	3	3.75	2.75	50.5995	0	4.2735
	38	17	4.25	1.25	45.6648	0	5.1444
Averages			3.68333333	1.56666666	48.4328	0.16223	4.80274
Pure Rectilinear					63.4749	0	3.8508
Pure Circular					55.7989	5.2989	6.4702
Pure Speed Blend					39.7704	0	6.3923
Pure Distance Blend					44.0568	0	7.2996
Top Overall Performer	10	12	3.5	1	45.4565	0	5.3983
	38	17	4.25	1.25	45.6648	0	5.1444

Figure 53: Summary of NRL to Van Ness Bearing Rate Blending

For this route, the top performing parameter sets were more varied. All of the functions appeared to produce similar results, but the logarithmic function routinely produced swarm behavior that maintained the UAVs closer to the convoy while sacrificing a minimal amount of steering energy. All of the top performing sets out performed the pure rectilinear and circular

control, however, the top performing speed and distance blends surpassed the performance of the bearing rate blending method for this GPS route.

The results of the bearing rate blending simulations on the Van Ness Street to USNA route are summarized in the table in Figure 54.

VN to USNA Blending Function	Trial	Times a Global Min	Max Bearing Rate (deg/sec)	Min Bearing Rate (deg/sec)	Convoy %	Collision %	Steering %
Linear	1	27	3.25	1	35.8175	0	6.3673
	10	8	3.5	1	36.1076	0	6.2503
	75	1	5.25	1.5	48.3833	0	4.5704
Exponential	10	36	3.5	1	44.3247	0.15138	5.5879
Hyperbolic Tangent	4	35	3.25	1.75	36.8506	0	6.2523
	52	1	4.5	2.5	45.7361	0.24933	5.0738
Logarithmic	1	32	4	2	26.3571	0	6.717
	64	3	3.25	1.5	29.3231	0.18997	5.8553
	74	1	3.25	1	34.4203	0	5.2418
Averages			3.75	1.47222222	37.4800	0.06563	5.76845
Pure Rectilinear					60.1059	0	4.8993
Pure Circular					24.1143	0.48679	9.8864
Pure Speed Blend					22.2784	0	9.4917
Pure Distance Blend					27.295	0.071238	9.2865
Top Overall Performer	32	1	4	2	26.3571	0	6.717

Figure 54: Summary of Van Ness to USNA Bearing Rate Blending

This route had multiple bearing rate blends as top performers for each function, but was not as widely distributed as the NRL to VN route. The performance even between the same function also varied considerably more than in the speed or distance blending results. This reinforces the idea that bearing rate blending method is much more sensitive to the blending function and the saturation limits imposed on the function. Unlike in the distance or speed blending methods, there doesn't appear to be any correlation to determine the best saturation limits for the bearing rate functions. The minimum and maximum bearing rates don't appear to

have been effected by variables in the route such as convoy speed or initial positions. Also, there does not appear to be any correlation between these limits and the specific functions. It seems that these values can vary greatly and this blending method is not sensitive to certain ranges of the saturation limits. This may be due to the somewhat crude nature of the bearing rate variable itself, which is also reflected in the wide range of performance across the different bearing rate blending functions.

For this route, the bearing rate blend outperformed pure rectilinear control as well as pure distance blending, but it failed to outperform pure circular or the speed blending top performance on the same route. This is most likely due to the fact that in this route, the convoy moves at a much slower pace, which results in pure circular control being highly effective. Also, measuring the speed of the convoy is a much more accurate variable to determine how to transition between the two forms of control. Bearing rate blending attempts to combine speed and distance into a single measurement, but it is not as strong a measurement as the convoy speed. While the bearing rate is not as precise a measurement as convoy speed or range, this variable proves to be an effective way to blend the two forms of the control law when the convoy is progressing at a slow rate.

For all three of the routes, the logarithmic bearing rate was the top performing blend. In most cases, the several logarithmic trials that were the top performing logarithmic blends out performed the other bearing rate blending functions. Based on this evidence, a logarithmic function appears to be the most effective bearing rate blending function. The logarithmic function has an initially fast transition to rectilinear control, but then levels out to full rectilinear control. This performs well for bearing rate blending because the bearing rate also rapidly changes from low to high, as seen in the bearing rate plot above. Thus, the logarithmic function

compliments the characteristics of the blending rate variable. As the bearing rate suddenly spikes as the UAVs approach and pass the convoy, the logarithmic function is able to quickly transition between rectilinear and circular control in an almost binary manner. This quick transition speed, which proved to be ineffective in speed blending, is beneficial to handle the spikes in bearing rate.

From this limited data, it is not possible to determine which of the three blending methods is superior. As shown in this analysis, adding additional variables to base the control law transition off of could increase swarm performance. When limitations were seen in the speed blending, a distance component was added that in effect created a blending of a blending when it combined speed blending with pure rectilinear control. While adding multiple layers of blending with additional variables might increase swarm performance, there is no indication that this increase would be significant when compared to the results from this research. What is clear, is that each blending method as its own unique mission which it would be well suited. If the convoy is going to be moving at a slow pace and the UAVs are going to start at a close distance, less than two miles, speed blending would be appropriate. However, if the convoy is going to be varying speeds widely and the UAVs start out a significant distance from the convoy, distance blending is most effective. Lastly, if operational security is the priority for the mission, and performance can be slightly sacrificed, bearing rate blending should be utilized. Specifically, a blending function appears to perform routinely well, and this form of blending can still give reasonable performance and keep the UAVs well within sensor range. If the mission requires the UAVs to be as close as possible to the convoy though, a distance or speed blending method might be preferred.

11. Implementations of Research and Future Work

This research served as the crucial link between the development and mathematical proof of the UMD control algorithm and potential implementation of this algorithm in field tests. While the control law had been proven to converge in mathematical proofs as well as basic MATLAB simulations, there was no previous research concerning the effects of the specific parameters on overall swarm behavior. This research analyzed the algorithm using realistic convoy routes and more advanced vehicle dynamics to study the effectiveness of using UAV swarms to provide urban convoy escort. For the pure rectilinear and circular forms of the control law, a strong understanding of the relationship between the magnitudes of the parameters and their effect on the swarm's performance was determined. This allowed for improved performance of the pure control forms when simulated in their respective ideal scenarios.

From the pure control form analysis came the development of several blending methods. At first, a blending method based on the speed of the convoy was evaluated. This method proved to be very effective when compared to the pure forms of the control law. However, this method assumed that the UAV's initial locations were relatively close (less than 2 miles) to the convoy's initial location. To improve upon the blending strategy, a method was examined in which the range of each UAV to the convoy was also included as a variable. For the distance blending method, the UAVs would use pure rectilinear control when outside of a certain user specified range. The blending function would then combine rectilinear control with speed blending control as the UAVs approached the convoy. When the UAVs were within a certain user specified minimum distance from the convoy, only speed blending control was used. With the addition of the UAV to convoy range component, the blending was improved. Swarm performance was no longer dependant on initial conditions. An unexpected benefit of the

distance variable addition was the ability to dampen sudden convoy accelerations. The distance blending combined pure rectilinear control with speed blending control. Contained in the speed blending was a portion of pure rectilinear control as well. Thus, distance blending utilized an increased amount of rectilinear control, which allowed the UAV swarm to respond faster to changes in convoy acceleration. As the convoy speed increased, not only would speed blending utilize more rectilinear control, but as the range between the UAVs and the convoy increased, the distance blending would utilize more pure rectilinear control as well.

Speed and distance blending methods used very accurate variables as a basis for transitioning between the two forms of swarm control supported by the UMD algorithm. This resulted in improved swarm performance when compared to the pure control forms and is a practical control strategy. However, these strategies rely on the exact convoy location expressed in a GPS string that is transmitted by the convoy. While this results in strong swarm performance, it is also a security risk in today's operations. Thus, a blending method which would not rely on GPS positions transmitted by the convoy was evaluated. By computing the relative bearing rate of the convoy from each UAV, it was possible to blend rectilinear and circular control to produce swarm behavior which generally improved upon simulations using only one of the pure control forms. Since the bearing rate calculation was not as precise as convoy speed and range, the bearing rate method was not as effective as the speed and distance blending strategies used earlier. Nevertheless, bearing rate blending proved to be a secure strategy for swarm control.

While these results help further the implementation of the UMD control law into real UAVs for future field tests, the process which has been used to evaluate the algorithm can be beneficial to operators in theatre right now. By using the blending strategies analyzed in this

research in conjunction with the parameter values that are most effective for the convoy escort scenario, operators can simulate their own convoy missions. They have the ability to "plug in" the convoy GPS coordinates as well as variables specific to their UAV platforms such as sensor range, speed, and fuel load in order to analyze an accurate representation of their convoy missions. This will allow them to asses the amount of coverage their UAV swarm will be able to provide to the convoy and the amount of time the UAVs will be available based on fuel limitations. If the mission requires more security than the UAVs are simulated to provide, there could be justification for altering the mission or, more likely, getting a more capable UAV platform for the swarm. If there are specific locations where the mission planner deems it essential to have UAV coverage, these waypoints can be loaded into the simulator and the UAVs can ignore the convoy and process only to these specified waypoints. This reduces overall fuel consumption and increases the accuracy of the UAV coverage. Lastly, based on this research, the mission planner has a foundation of knowledge for the effects of the swarm size on the parameter sets used for the control. This research concluded that changing these gain values does not increase overall swarm performance drastically when the convoy reduces in size. However, if the convoy size increases due to the late launch of a UAV, the mission planner has justification that the parameter values for the control law should be modified to improve swarm performance.

This research not only progressed the investigation of the UMD control algorithm through realistic simulations, it also made the tools available for similar analysis available at USNA. SIMDIS along with the multi-vehicle simulator is installed and set up on multiple machines at USNA along with a detailed guide for basic set-up and operation of these tools. MATLAB scripts are available to convert basic vehicle data into the .asi file that is used by SIMDIS. A Garmin GPS is also available for students to capture position data for

implementation into the simulator. These tools are already being used in current Systems Department design projects. Lastly, a cumulative analysis process has been developed for USNA and NRL to utilize for future investigation of multi-vehicle control algorithms. This will allow for a more efficient and directed analysis of potential control algorithms which will expedite the time it takes for the control laws to go from mathematical proofs to implementation in real vehicles.

Future work on the UMD control law can now be focused on the implementation of the algorithm in real vehicles. This research identified the limitations and strengths of the algorithm so that future researchers have a basic understanding of the performance of a swarm utilizing this control law. While the control law is ultimately limited by the dynamics of the UAV platform, the control law is also sensitive to the parameter values. Increasing a specific parameter such as α will result in the vehicles maneuvering closer to each other and the convoy, but at the cost of increased collisions and potentially higher amounts of steering energy. This research identifies these relationships and limitations so that future researchers have an accurate idea about the effects of the specific behaviors the algorithm attempts to model. In the future, less work will need to be focused on the specific parameter values and more will be done with the implementation of the algorithm into real vehicles. This will be the true test of performance for the control law. Also, evaluating the effects of communications degradation between the UAVs and convoy as well as GPS drop out may be beneficial. With the role of electronic attack in mind, limiting the communications between vehicles is important for security purposes. Identifying the frequency of communications necessary to maintain swarm performance would be an important area of research.

In conclusion, this research provided a strong foundation of data and conclusions to support further investigation and implementation of the UMD algorithm. Depending on the scenario, distance or bearing rate blending is shown to be an effective way to maintain sensor coverage around the convoy with a UAV swarm. Future investigations can delve even farther by simulating communications degradation and GPS dropout and evaluating their effects on swarm performance. Identifying the minimum communications frequency that still enables acceptable swarm performance is also an important area of research for defense from electronic attack. Lastly, this research organized a cumulative evaluation process for future algorithms and enables the tools for this research to be implemented at USNA for future research projects.

12. Endnotes

[1] J. Cheng, W. Cheng, Nagpal, "Robust and Self-repairing Formation Control For Swarms Of Mobile Agents," *National Conference on Artificial Intelligence (AAAI '05),* July 2005.

[2] L. Barnes, W. Alvis, M. Fields, K. Valavanis, W. Moreno, "Heterogeneous Swarm Formation Control Using Bivariate Normal Functions to Generate Potential Fields," *IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications,* pp. 85-94, 15-16 June 2006.

[3] Z. Cao, M. Tan, S. Wang, Y. Fan, B. Zhang, "The optimization research of formation control for multiple mobile robots," *Proceedings of the 4th World Congress on Intelligent Control and Automation,*, vol.2, pp. 1270-1274, 2002.

[4] S. Zelinski, T.J. Koo, S. Sastry, "Optimization-based formation reconfiguration planning for autonomous vehicles," *IEEE International Conference on Robotics and Automation,* vol.3, pp. 3758-3763, September 2003.

[5] "February Iraqi Election Photos," *CFLCC Today,* USARCENT, Feb. 2005. [Website], Available: http://www.arcent.army.mil/cflcc_today/2005/february. [Accessed: Feb 2, 2008].

[6] H.V.D. Paranak, "Go to the ant: Engineering principles from natural multi-agent systems", *Annals of Operations Research,* 1997.

[7] C. Reynolds, "Flocks, herds and schools: A distributed behavioral Model," Computer Graphics, vol. 21, no. 4, pp.25-34, July 1987.

[8] W. Justh, V. Kowtha, "Biologically inspired models for swarming," *Evolutionary and Bio-inspired Computation: Theory and Applications,* vol. 6563, April 2007.

[9] X. Tu and D. Terzopoulos, "Artificial fishes: Physics, locomotion, perception, behavior," *Proc. SIGGRAPH 94 Conf.* Orlando, FL, pp. 43-50, July 1994.

[10] P. K. C. Wang, "Navigation strategies for multiple autonomous robots moving in formation," *J. Robot. Syst.*, vol. 8, no. 2, pp. 177-195, 1991

[11] M. Mataric, "Designing emergent behaviors: From local interactions to collective intelligence," *Proc. Int. Conf. Simulation of Adaptive Behavior: From Animals to Animats 2*, pp. 432-441, 1992.

[12] E.W. Justh, P.S. Krishnaprasad, "Equilibria and steering laws for planar formations" *Systems and Control Letters*, vol. 52, no. 1, pp. 25-38, 2004.

[13] D. W. Gage, "Command control for many-robot systems," *Unmanned Syst. Mag.*, vol. 10, no. 4, pp. 28-34, 1992.

[14] C. J. McCook, J.M. Esposito, "Flocking for Heterogeneous Robot Swarms: A Military Convoy Scenario," *Thirty-Ninth Southeastern Symposium on System Theory,* pp.26-31, March 2007.

13. Bibliography

C. J. McCook, J.M. Esposito, “Flocking for Heterogeneous Robot Swarms: A Military Convoy Scenario,” Thirty-Ninth Southeastern Symposium on System Theory, pp.26-31, March 2007.

C. Reynolds, “Flocks, herds and schools: A distributed behavioral Model,” Computer Graphics, vol. 21, no. 4, pp.25-34, July 1987.

D. W. Gage, “Command control for many-robot systems,” Unmanned Syst. Mag., vol. 10, no. 4, pp. 28-34, 1992.

E.W. Justh, P.S. Krishnaprasad, “Equilibria and steering laws for planar formations” Systems and Control Letters, vol. 52, no. 1, pp. 25-38, 2004.

“February Iraqi Election Photos,” CFLCC Today, USARCENT, Feb. 2005. [Website], Available: http://www.arcent.army.mil/cflcc_today/2005/february. [Accessed: Feb 2, 2008].

H.V.D. Paranak, “Go to the ant: Engineering principles from natural multi-agent systems”, Annals of Operations Research, 1997.

J. Cheng, W. Cheng, Nagpal, “Robust and Self-repairing Formation Control For Swarms Of Mobile Agents,” National Conference on Artificial Intelligence (AAAI '05), July 2005.

L. Barnes, W. Alvis, M. Fields, K. Valavanis, W. Moreno, “Heterogeneous Swarm Formation Control Using Bivariate Normal Functions to Generate Potential Fields,” IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications, pp. 85-94, 15-16 June 2006.

M. Mataric, “Designing emergent behaviors: From local interactions to collective intelligence,” Proc. Int. Conf. Simulation of Adaptive Behavior: From Animals to Animats 2, pp. 432-441, 1992.

P. K. C. Wang, “Navigation strategies for multiple autonomous robots moving in formation,” J. Robot. Syst., vol. 8, no. 2, pp. 177-195, 1991

S. Zelinski, T.J. Koo, S. Sastry, “Optimization-based formation reconfiguration planning for autonomous vehicles,” IEEE International Conference on Robotics and Automation, vol.3, pp. 3758-3763, September 2003.

W. Justh, V. Kowtha, “Biologically inspired models for swarming,” Evolutionary and Bio-inspired Computation: Theory and Applications, vol. 6563, April 2007.

X. Tu and D. Terzopoulos, “Artificial fishes: Physics, locomotion, perception, behavior,” Proc. SIGGRAPH 94 Conf. Orlando, FL, pp. 43-50, July 1994.

Z. Cao, M. Tan, S. Wang, Y. Fan, B. Zhang, “The optimization research of formation control for multiple mobile robots," Proceedings of the 4th World Congress on Intelligent Control and Automation,, vol.2, pp. 1270-1274, 2002.

14. Appendix

1. MATLAB Simulation Script

```
%=========================================================================
% File Name    : mvscript_wpt_blend_transition_pervcl.m
% Authors      : W. Selby
% Date         : 19-May-2008
% Description  : Simulates UMD control algorithm and plots the output
           for vehicles following a set of waypoints moving based
on each vehicle's distance to the waypoint and
switching at a specific waypoint and blending from
rectilinear to circular.
% Inputs:      : None
% Outputs:     : None
% Requirements : None
% Revisions    : None
% =========================================================================

%% Initialization
clc; clear all; close all;

tfin  = 2000.0;  %run time
dt    = 0.1;     %step time
index = 1.0;     %index
n     = 7;       %number of vehicles, must be >1 b/c waypoint is a vehicle
vel   = 1.0;     %velocity
eta     =  1/(n-1);       %perpendicular to baseline
mu     =  1/(n-1);       %heading alignment
alpha   =  1/(n-1);       %equidistance
ro      =  10.0;     %minimum distance (meters)
rwpt    =  10.0;       %Proximity to waypoint distance
sumterm_rect =  0.0;      %Sum of the control terms
sum_wpt_rect =  0.0;      %steering command for waypoint
sumterm_circ =  0.0;      %Sum of the control terms
sum_wpt_circ =  0.0;      %steering command for waypoint
umax    =  0.5;      %Maximum steering command
trans   =    0.0;      %%Percentage of circular control to use 0<trans<1
tdmax   =    300.0;    %%Distance from waypoint to begin transition
tdmin   =    100.0;    %%Distance from waypoint to end transition

for(i=1:n)           % Set all waypoints to first point
    place(i)=1;
    counter(i)=0;
end

%% Vehicle initial positions
if(1)            %Input same positions for every simulation
    r(1,7)=-10;
    r(2,7)=-2;
    r(1,2)=20;
    r(2,2)=11;
    r(1,3)=0;
    r(2,3)=14;
    r(1,4)=-15;
    r(2,4)=-22;
```

```
    r(1,5)=27;
    r(2,5)=0;
    r(1,6)=0;
    r(2,6)=27;
    theta(7)= 1.45;
    theta(2)= -pi;
    theta(3)= 2.1;
    theta(4)= 5.75;
    theta(5)= 3.3;
    theta(6)= 6.0;

    for(j=1:n)            %Set all initial conditions to random numbers
        u(j)=0.0;
        for(i=1:n)
            rsep(j,i)=0.0;
        end
    end

else
    for(j=1:n)            %Set all initial conditions to random numbers
        u(j)=0.0;           %steering control
        r(1,j)=50*randn(1);    %x values
        r(2,j)=50*randn(1);    %y values
        theta(j)=randn(1);  %thetas
        for(i=1:n)
            rsep(j,i)=0.0;
        end
    end
end

%% Simulation loop
for(t=0:dt:tfin)

    %% Waypoint vehicle
    wpt=[ 150, 100;        %%Position of Waypoint 1
        450, -50;          %%Position of Waypoint 2…
        200, -50;
        300, 0
        50, 150
        200, -50];

    %% Integration/vehicle dynamics

    for(j=2:n)                                  %% Not for Waypoint
        theta(j)  = theta(j)+u(j)*dt;
        r(1,j)    = vel*cos(theta(j))*dt+r(1,j);
        r(2,j)    = vel*sin(theta(j))*dt+r(2,j);
    end

    %% Control Per Vehicle
    for(j=1:n)
        xj= [cos(theta(j)); sin(theta(j)) ];
        yj_temp= cross([0 0 1], [xj(1,1) xj(2,1) 0]);
        yj = yj_temp(1:2);
```

```
if(place(j)==5)         %%Use transition control
    for(k=1:n)
        if(k~=j)
            if(k==1)            %%Computes waypoint properties

                %% Make sure waypoint exists
                if(place(j)<=size(wpt,1))

                %% Update waypoint coordinates
                    r(1,1)= wpt(place(j),1);
                    r(2,1)= wpt(place(j),2);
                end
                rjk = [ r(1,j)-r(1,1); r(2,j)-r(2,1) ];
                rsep(j,k) = norm(rjk);

                %% Direction to waypoint
                theta(1)=pi+atan2(rjk(2,1),rjk(1,1));
                xk= [cos(theta(1)); sin(theta(1))];
                runit = rjk/norm(rjk);

                %%Non-reduced waypoint control
                sum_wpt_circ = -eta*[dot(runit,xj)]*[dot(runit,yj)]
                -alpha*[1-(ro/norm(rjk))^2]*[dot(runit,yj)];

                %%Non-reduced waypoint control
                sum_wpt_rect = -eta*[dot(runit,xj)]*[dot(runit,yj)]
                -alpha*[1-(ro/norm(rjk))^2]*[dot(runit,yj)]
                +dot(mu*xk,yj);

                %%Tells when to switch to next waypoint based on
                distance
                if(norm(rjk)<=2*rwpt)

                %% Time delay until waypoint switch
                    counter(j) = counter(j)+1;
                end

                %%Delay is over, switch to next waypoint
                if(counter(j)==3000)
                    place(j)=place(j)+1;
                end

                %% Transition Parameter Calculation
                if(norm(rjk)>tdmax)
                    trans=0;
                end
                if(norm(rjk)<tdmin)
                    trans=1;
                end
                if(norm(rjk)<tdmax && norm(rjk)>tdmin)
                   trans=1-(1/(tdmax-tdmin))*(norm(rjk)-tdmin);
                end
            end
            rjk = [ r(1,j)-r(1,k); r(2,j)-r(2,k) ];
            rsep(j,k) = norm(rjk);
            xk= [cos(theta(k)); sin(theta(k))];
```

```
                runit = rjk/norm(rjk);

                sumterm_circ= -eta*[dot(runit,xj)]*[dot(runit,yj)]
                    -alpha*[1-(ro/norm(rjk))^2]*[dot(runit,yj)];

                sumterm_rect= -eta*[dot(runit,xj)]*[dot(runit,yj)]
                    -alpha*[1-(ro/norm(rjk))^2]*[dot(runit,yj)]...
                    +dot(mu*xk,yj);

                %% Transition Control Calculation
                if(trans==0)     %%Use rectilinear
                    u(j)=(1/n)*(sumterm_rect+u(j))+sum_wpt_rect;
                end
                if(trans==1)     %%Use circular
                    u(j)=(1/n)*(sumterm_circ+u(j))+sum_wpt_circ;
                end
                if(trans<1 && trans>0)      %%Blend Control
                    u(j)=(1/n)*[ (1-trans)*sumterm_rect +
                    trans*sumterm_circ + u(j)]                          +(1-
                    trans)*sum_wpt_rect + trans*sum_wpt_circ;
                end
                if(u(j)>umax)
                    u(j)=umax;
                elseif(u(j)<-umax)
                    u(j)=-umax;
                end
            end
        end

    else            %% Use rectilinear Control
        for(k=1:n)
            if(k~=j)
                if(k==1)
                 %%Computes waypoint properties

                    %% Make sure waypoint exists
                    if(place(j)<=size(wpt,1))
                    %% Update waypoint coordinates
                        r(1,1)= wpt(place(j),1);
                        r(2,1)= wpt(place(j),2);
                    end
                    rjk = [ r(1,j)-r(1,1); r(2,j)-r(2,1) ];
                    rsep(j,k) = norm(rjk);
                    %% Direction to waypoint
                    theta(1)=pi+atan2(rjk(2,1),rjk(1,1));
                    xk= [cos(theta(1)); sin(theta(1))];
                    runit = rjk/norm(rjk);

                    %%Non-reduced waypoint control
                    sum_wpt_rect = -eta*[dot(runit,xj)]*[dot(runit,yj)]
                    -alpha*[1-(ro/norm(rjk))^2]*[dot(runit,yj)]
                    +dot(mu*xk,yj);

                    %%Tells when to switch to next waypoint based on
                    distance
                    if(norm(rjk)<=2*rwpt)
```

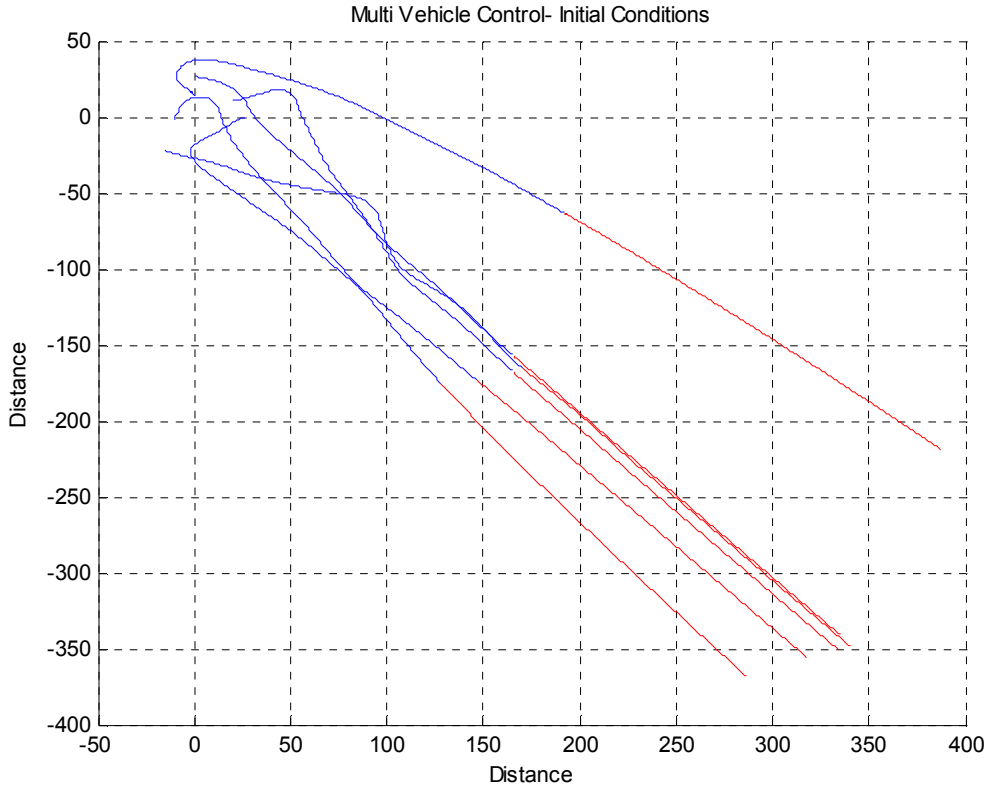
```
                        %% Move to next waypoint
                            place(j)=place(j)+1;
                        end
                    end
                    rjk = [ r(1,j)-r(1,k); r(2,j)-r(2,k) ];
                    rsep(j,k) = norm(rjk);
                    xk= [cos(theta(k)); sin(theta(k))];
                    runit = rjk/norm(rjk);

                    sumterm_rect= -eta*[dot(runit,xj)]*[dot(runit,yj)]
                        -alpha*[1-(ro/norm(rjk))^2]*[dot(runit,yj)]
                        +dot(mu*xk,yj);

                    u(j)=(1/n)*(sumterm_rect+u(j))+sum_wpt_rect;
                    if(u(j)>umax)
                        u(j)=umax;
                    elseif(u(j)<-umax)
                        u(j)=-umax;
                    end
                end
            end
        end
    end

    %% Logging
    for(j=1:n)
        LV_theta(index,j) = theta(j);
        LV_rx(index,j) = r(1,j);
        LV_ry(index,j) = r(2,j);
        LV_time(index) = t;            %Logs time
        LV_u(index,j)  = u(j);         %Log Steering Control
        for(i=1:n)
            LV_rsep(index,j,i)  = rsep(j,i);
        end
    end
    index=index+1;
end

%% Plotting
% Initialize the figure
if(isempty(findobj('UserData',gcb)))
    DIPFigure = figure('MenuBar','figure',...
        'NumberTitle','off','Resize','off','Name','Vehicle Control
Animation');
    set(DIPFigure,'UserData',gcb);
    set(DIPFigure,'DoubleBuffer','on');
    set(gca,'fontsize',8);
    grid on; hold on;
    title('Multi Vehicle Control - Following Waypoints');
    xlabel('Distance');
    ylabel('Distance');
end
if(1)
    %%Plot Path
    if(1)
```

```
        for(j=2:n)
            plot(LV_rx(:,j),LV_ry(:,j),'b')
            vehicle(j) = rectangle('position',[LV_rx(size(LV_rx,1),j)
LV_ry(size(LV_rx,1),j)-.5*2 2 2],'curvature',...
                [1 1],'linewidth',.01,'facecolor','r');
        end
        for(j=1:n)
            vehicle(j) = rectangle('position',[LV_rx(size(LV_rx,1),j)
LV_ry(size(LV_rx,1),j)-.5*2 2 2],'curvature',...
                [1 1],'linewidth',.01,'facecolor','r');
            for(j=1:size(wpt,1))
                vehicle(j) = rectangle('position',[wpt(j,1) wpt(j,2)-.5*2 2
2],'curvature',...
                    [1 1],'linewidth',.01,'facecolor','r');
            end
        end
    end
    %% Plot Steering Control u
    if(1)
        for(j=1:n)
            figure(2);
            title('Steering Control Output');
            xlabel('Time');
            ylabel('Steering Control');
            hold on; grid on;
            plot(LV_time,LV_u(:,j));
        end
    end
    %%Plot Vehicle Separation Distances
    if(1)
        figure(3);
        title('Vehicle Separation Distances');
        xlabel('Time');
        ylabel('Distance');
        hold on; grid on;
        line(0:tfin,ro,'Color','r');
        for(j=1:n)
            for(i=1:n)
                plot(LV_time,LV_rsep(:,j,i));
            end
        end
    end
else
%% Can see vehicles move
    for(h=1:n)
        vehicle(h) = rectangle('position',[ r(1,j) r(2,j) .2
.2],'curvature',...
            [1 1],'linewidth',.01,'facecolor','r');
    end
    for(i=1:index-1)
        for(j=1:n)
            set(vehicle(j),'position',[LV_rx(i,j) LV_ry(i,j)-.5*.2 .2 .2]);
            plot(LV_rx(i,j), LV_ry(i,j),'b--','LineWidth',10);
            pause(.1*dt)
        end
    end
end
```

2. Parameter Modification Plots

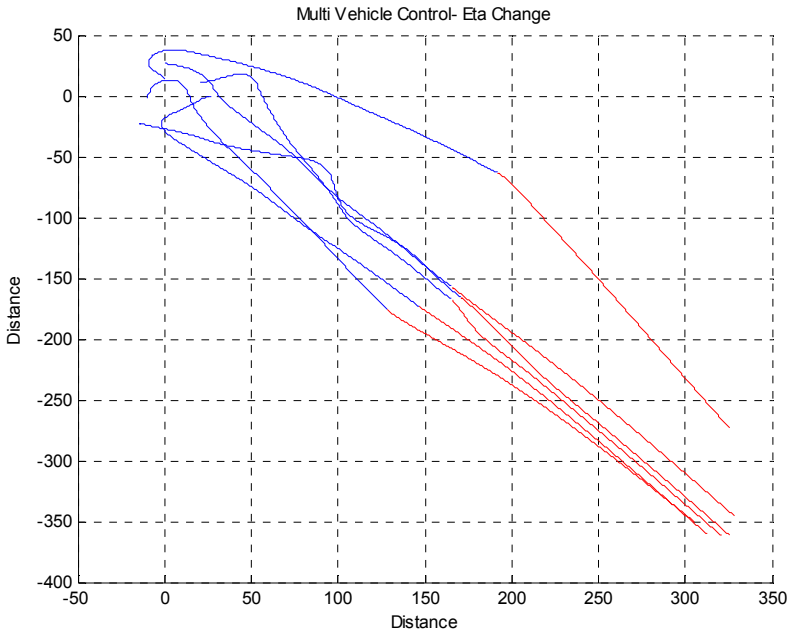
2.1. Plot of Initial Conditions

	Eta	Mu	Alpha	Vel	Ro	Tfin	dt	n	Filename
Initial	0.9	0.5	0.2	1	20	500	0.1	6	mvscript_gainmod.m
Final									

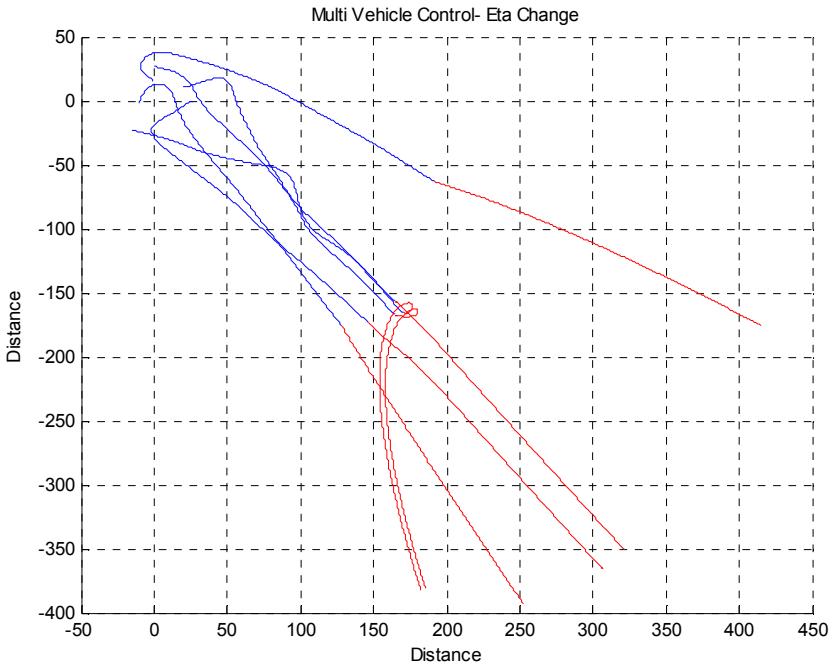


2.2. η Modified

	Eta	Mu	Alpha	Vel	Ro	Tfin	dt	n	Filename
Initial	0.9	0.5	0.2	1	20	500	0.1	6	mvscript_gainmod.m
Final	0.09								

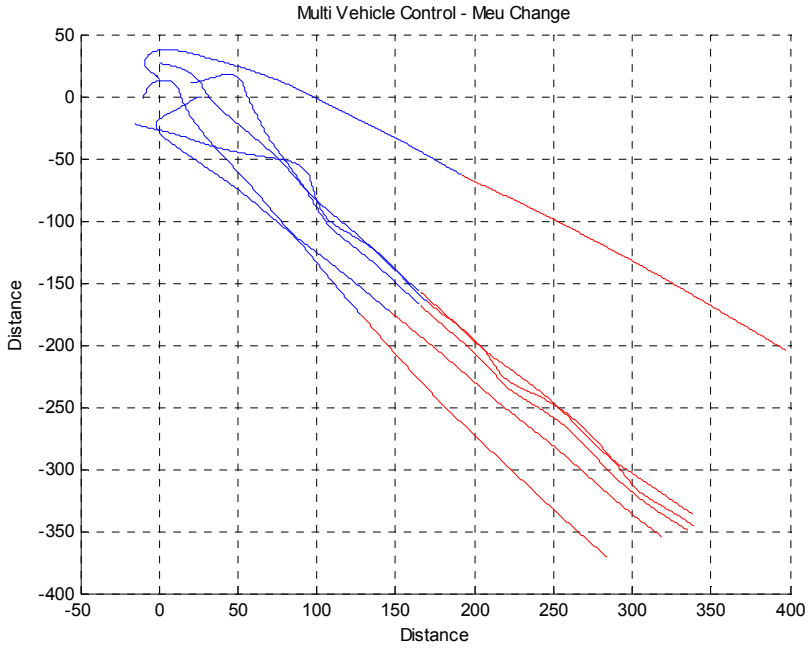


	Eta	Mu	Alpha	Vel	Ro	Tfin	dt	n	Filename
Initial	0.9	0.5	0.2	1	20	500	0.1	6	mvscript_gainmod.m
Final	9								

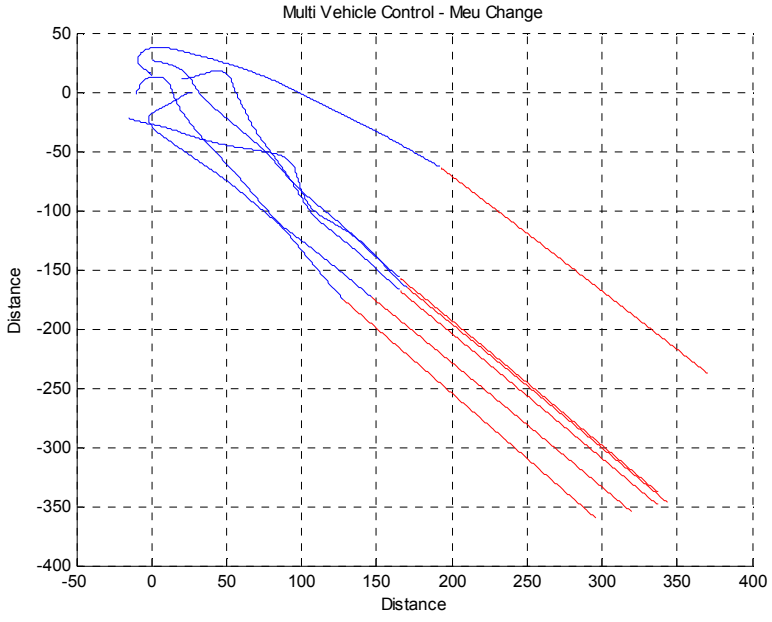


2.3. μ Modified

	Eta	Mu	Alpha	Vel	Ro	Tfin	dt	n	Filename
Initial	0.9	0.5	0.2	1	20	500	0.1	6	mvscript_gainmod.m
Final		0.05							

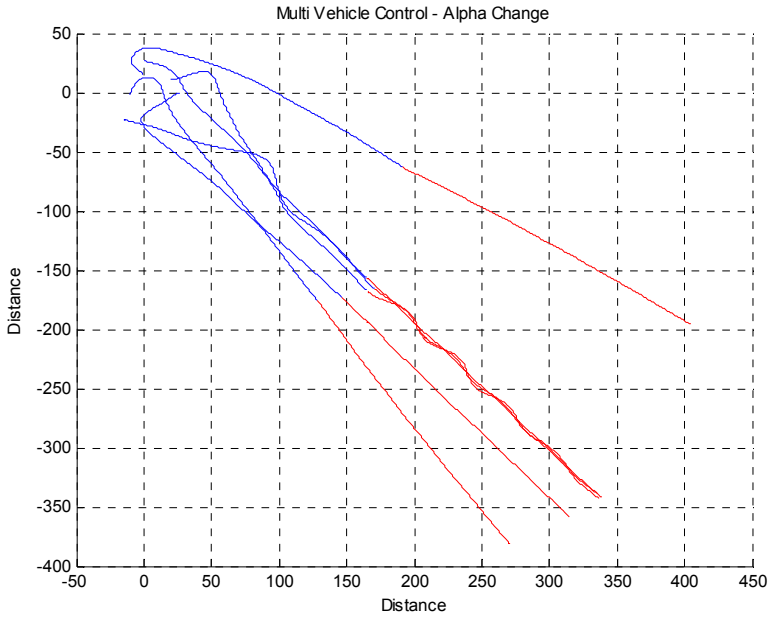


	Eta	Mu	Alpha	Vel	Ro	Tfin	dt	n	Filename
Initial	0.9	0.5	0.2	1	20	500	0.1	6	mvscript_gainmod.m
Final		5							

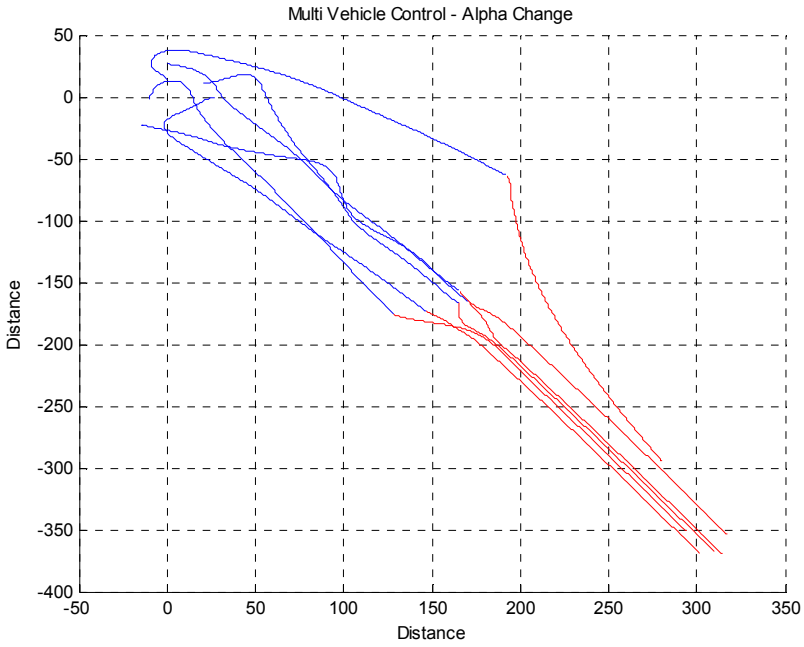


2.4. α Modified

	Eta	Mu	Alpha	Vel	Ro	Tfin	dt	n	Filename
Initial	0.9	0.5	0.2	1	20	500	0.1	6	mvscript_gainmod.m
Final			0.02						

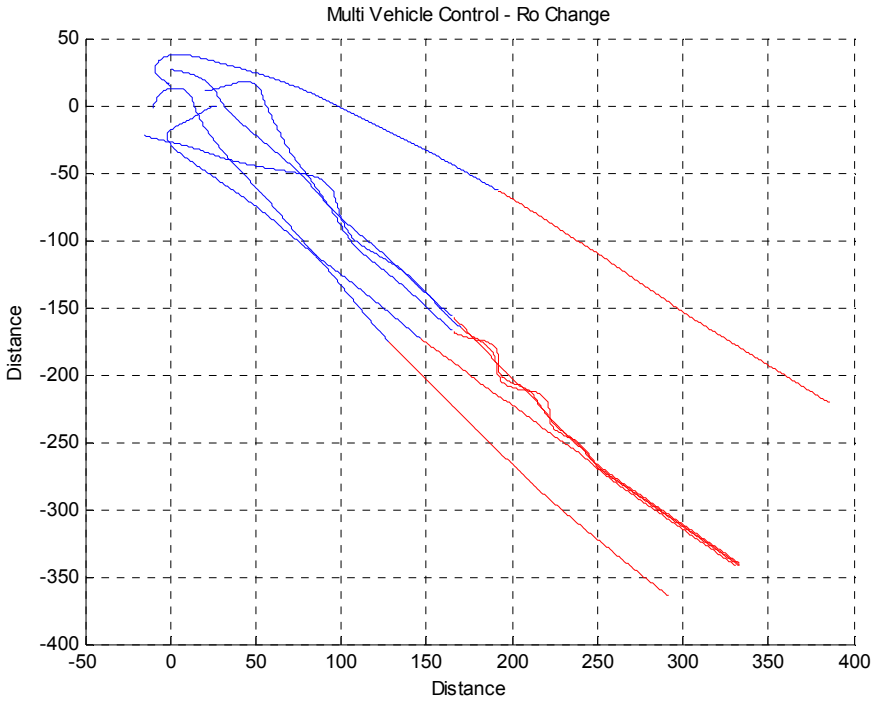


	Eta	Mu	Alpha	Vel	Ro	Tfin	dt	n	Filename
Initial	0.9	0.5	0.2	1	20	500	0.1	6	mvscript_gainmod.m
Final			2						

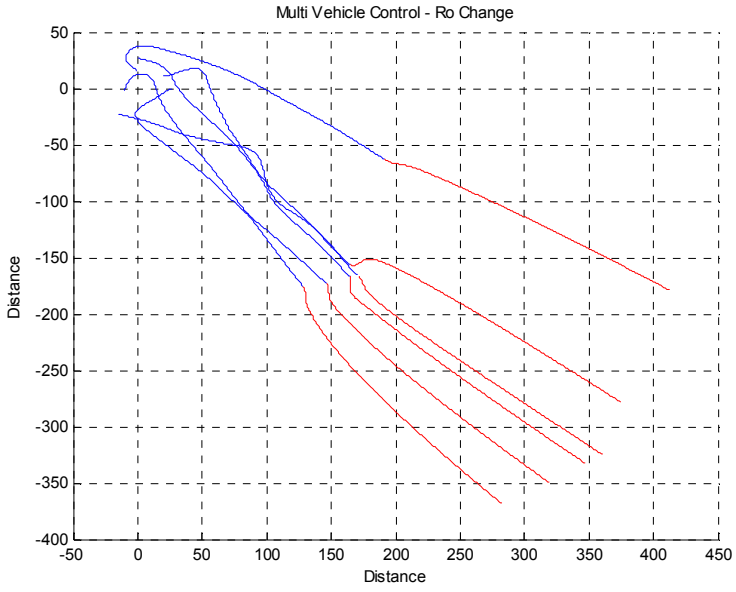


2.5. r_0 Modified

	Eta	Mu	Alpha	Vel	Ro	Tfin	dt	n	Filename
Initial	0.9	0.5	0.2	1	20	500	0.1	6	mvscript_gainmod.m
Final					0.2				



	Eta	Mu	Alpha	Vel	Ro	Tfin	dt	n	Filename
Initial	0.9	0.5	0.2	1	20	500	0.1	6	mvscript_gainmod.m
Final					80				



3. JAVA Simulation of UMD Control Algorithm

3.1. Vehicle.java

```
package advancedControl;

public class Vehicle {

	//Simulation Paramters
	private double dt = 0.1;
	public static double vel = 1.0;

	//Vehicle  Orientation
	public Vector Rxy = new Vector();
	public double theta;
	public double u;
	public Vector x = new Vector();
	public Vector y = new Vector();
	public Vector z = new Vector(0.0, 0.0, 1.0);

	public Vehicle() {	} //Constructor

	public void newPosition() {   //Integration and progression
		this.theta = theta + u * dt;
		this.Rxy.elem[0] = Rxy.elem[0] + vel * Math.cos(theta) *
	dt;
		this.Rxy.elem[1] = Rxy.elem[1] + vel * Math.sin(theta) *
	dt;
		this.setPosit(Rxy, theta);
	}
	//Set position for Waypoints
	public void setPosit(double newRx, double newRy){
	this.Rxy.elem[0]= Rxy.elem[0];
		this.Rxy.elem[1]= Rxy.elem[1];
	}

	public void setPosit(Vector Rxy, double newTheta) {		//Update
position and heading vectors
		this.Rxy.elem[0]= Rxy.elem[0];
		this.Rxy.elem[1]= Rxy.elem[1];
		this.theta =newTheta;

		this.x.elem[0] = Math.cos(theta);
		this.x.elem[1] = Math.sin(theta);
		this.x.elem[2] = 0;

		this.y.cross(z, x);
	}

	public void setU(double u_in) {   //Set control limits
		if(u_in>.5)
			this.u = 0.5;
		if(u_in<-.5)
			this.u = -.5;
		if(u_in<=.5 && u_in>=-.5)
			this.u = u_in;
```

```
	}

}
```

3.2. Swarm.java

```
package advancedControl;

public class Swarm {

	// Vehicle Numbers
	public static int n = 6;
	public static double num = n;

	//Creates n number of new vehicles
	public Vehicle[] vehicle = new Vehicle[n];

	// Vehicle Initial Positions
	public Vector R1 = new Vector(-10,-2,0);
	public Vector R2 = new Vector(20,11,0);
	public Vector R3 = new Vector(0,14,0);
	public Vector R4 = new Vector(-15,-22,0);
	public Vector R5 = new Vector(27,0,0);
	public Vector R6 = new Vector(0,27,0);

	//Data Logging Variables
	public Vector LV_rx = new Vector();
	public Vector LV_ry = new Vector();



	public Swarm() {		// Constructor
		for (int i = 0; i < n; i++) {
			vehicle[i] = new Vehicle();
		}
	}
	//Set vehicle initial conditions (x,y,z) , theta
	public void initSwarm() {
		vehicle[0].setPosit(R1,1.45);
		vehicle[1].setPosit(R2, 0);
		vehicle[2].setPosit(R3, 2.1);
		vehicle[3].setPosit(R4, 5.75);
		vehicle[4].setPosit(R5, 3.3);
		vehicle[5].setPosit(R6, 6.0);
	}

	public void moveSwarm() {//Move all vehicles and update positions
		for (int i = 0; i < vehicle.length; i++) {
			vehicle[i].newPosition();
		}
	}

}
```

3.3. waypointVector.java

```
package advancedControl;
```

```
public class waypointVector extends Vector {

	public int delay;

	public waypointVector(double x, double y, double z, int delay){
		elem[0] = x;
		elem[1] = y;
		elem[2] = z;
		this.delay = delay;
	}
}
```

3.4. Waypoint.java

```
package advancedControl;
import java.lang.Math;
import gpsModule.*;

public class Waypoint extends Vehicle {

	// Waypoint properties
	double dist = 10; // Minimum distance to waypoint
	int numpts = 6; // Number of waypoints
	public int place = 0; // Current Waypoint location
	public int count = 0;

	public static final double PI = 3.141592653589793;



	// Waypoint Locations (x,y,z,flag)
	public waypointVector wpt1 = new waypointVector(150, 100, 0, 10000);
	public waypointVector wpt2 = new waypointVector(450, -50, 0, 5000);
	public waypointVector wpt3 = new waypointVector(200, -50, 0, 0);
	public waypointVector wpt4 = new waypointVector(300, 0, 0, 5000);
	public waypointVector wpt5 = new waypointVector(50, 150, 0, 0);
	public waypointVector wpt6 = new waypointVector(200, -50, 0, 0);

	// Waypoint Locations in a List
	public waypointVector[] wptList = new waypointVector[numpts];

	public Waypoint() { // Constructor
		wptList[0] = wpt1;
		wptList[1] = wpt2;
		wptList[2] = wpt3;
		wptList[3] = wpt4;
		wptList[4] = wpt5;
		wptList[5] = wpt6;
	}

	public Vector getPosition() { // Update Waypoint Location
		return wptList[place];
	}

	//Check distance to waypoint and move to next point
```

```
	public void checkAndSwitch(Vector R) {
		Vector rsep = new Vector();
		double length;
		rsep.elem[0] = R.elem[0] - wptList[place].elem[0];
		rsep.elem[1] = R.elem[1] - wptList[place].elem[1];
		rsep.elem[2] = 0;
		length = rsep.norm();
		if (wptList[place].delay>=0) { // if true- should delay
			if (length <= dist) {
				wptList[place].delay--;
				if (wptList[place].delay == 0) {
					if (place < wptList.length - 1) {
						place++;
					}
				}
			}
		} else {		//if false -move to next point
			if (length <= dist) {
				if (place < wptList.length - 1) {
					place++;
				}
			}
		}
	}
//Check distance to waypoint and move to next point
	public double distToWpt(Vector R) {
		Vector rsep = new Vector();
		double length;
		rsep.elem[0] = R.elem[0] - wptList[place].elem[0];
		rsep.elem[1] = R.elem[1] - wptList[place].elem[1];
		rsep.elem[2] = 0;
		length = rsep.norm();
		return length;
	}
// Calculate distance from Vehicle to waypoint
	public double dirToWpt(Vector R) {
		double newTheta;
		Vector rsep = new Vector();
		rsep.elem[0] = R.elem[0] - wptList[place].elem[0]; // x
		rsep.elem[1] = R.elem[1] - wptList[place].elem[1]; // y
		rsep.elem[2] = 0; // z
		rsep.divid(rsep.norm());
		newTheta = PI + Math.atan2(rsep.elem[1], rsep.elem[0]);
		return newTheta;
	}

	@Override
	public void newPosition() {
	} // Override movement of waypoints

}
```

3.5. Control.java

```
package advancedControl;

public class Control {

            // Gain Paramaters
            private static double eta = 1/(Swarm.num-1);
            static double alpha =1/(Swarm.num-1);
            static double mu = 1/(Swarm.num-1);
            static double eta_circ = .9;
            static double alpha_circ = .2;
            public static double ro = 10.0;

            //Create Waypoint Object
            Waypoint wpts = new Waypoint();

            // Control Calculation Variables
            double u;
            double norm;
            double sumterm;
            public Vector rjk = new Vector();
            public int delay = 400;

            public Control(){}      //Constructor


            public double calcControlRect(int vid, Swarm swarm) {
      //Rectilinear control
                  double tempu=0;
                  Vector tempkx = new Vector();
                        for (int k = 0; k < swarm.vehicle.length; k++) {
                              if (k != vid) {

this.rjk.elem[0] = swarm.vehicle[vid].Rxy.elem[0] -
swarm.vehicle[k].Rxy.elem[0];

this.rjk.elem[1] = swarm.vehicle[vid].Rxy.elem[1] -
swarm.vehicle[k].Rxy.elem[1];
                                          this.norm = rjk.norm();
                                          this.rjk.divid(rjk.norm());

tempkx.elem[0] = swarm.vehicle[k].x.elem[0] * mu;

tempkx.elem[1] = swarm.vehicle[k].x.elem[1] * mu;

tempkx.elem[2] = 0;


this.sumterm = -eta * Vector.dot(rjk, swarm.vehicle[vid].x)* Vector.dot(rjk,
swarm.vehicle[vid].y)- alpha* (1 - (ro / norm)*(ro / norm)) * Vector.dot(rjk,
swarm.vehicle[vid].y)+ Vector.dot(tempkx, swarm.vehicle[vid].y);
```

```
                        tempu = (1 / Swarm.num) * (sumterm + tempu);

                              }
                        }
                        return tempu;
                  }

            public double calcControlCirc(int vid, Swarm swarm) {
      //Circular Control
                  double tempu=0;
                        for (int k = 0; k < swarm.vehicle.length; k++) {
                              if (k != vid) {

this.rjk.elem[0] = swarm.vehicle[vid].Rxy.elem[0] -
swarm.vehicle[k].Rxy.elem[0];

this.rjk.elem[1] = swarm.vehicle[vid].Rxy.elem[1] -
swarm.vehicle[k].Rxy.elem[1];
                                          this.norm = rjk.norm();
                                          this.rjk.divid(rjk.norm());


this.sumterm = -eta_circ * Vector.dot(rjk, swarm.vehicle[vid].x)*
Vector.dot(rjk, swarm.vehicle[vid].y)- alpha_circ * (1 - (ro / norm)*(ro /
norm)) * Vector.dot(rjk, swarm.vehicle[vid].y);


                        tempu = (1 / Swarm.num) * (sumterm + tempu);

                              }
                        }
                        return tempu;
                  }

            public double calcWptControl(int vid, Swarm swarm) {
      //Waypoint Oriented Control
                  double tempu = 0;
                  Vector tempkx = new Vector();
                        wpts.getPosition();
                        wpts.checkAndSwitch(swarm.vehicle[vid].Rxy);

this.rjk.elem[0] = swarm.vehicle[vid].Rxy.elem[0]-
wpts.wptList[wpts.place].elem[0];

this.rjk.elem[1] = swarm.vehicle[vid].Rxy.elem[1]-
wpts.wptList[wpts.place].elem[1];

                        this.norm = rjk.norm();
                        this.rjk.divid(rjk.norm());

                        if (true){
            // Heading Control

wpts.theta = wpts.dirToWpt(swarm.vehicle[vid].Rxy);


tempkx.elem[0] = Math.cos(wpts.dirToWpt(swarm.vehicle[vid].Rxy)) * mu;
```

```
tempkx.elem[1] = Math.sin(wpts.dirToWpt(swarm.vehicle[vid].Rxy)) * mu;
                        tempkx.elem[2] = 0;
                        }
                        else{
            //No heading control
tempkx.elem[0] = swarm.vehicle[vid].x.elem[0] * mu;

tempkx.elem[1] = swarm.vehicle[vid].x.elem[1] * mu;
                            tempkx.elem[2] = 0;
                        }

                        this.sumterm = -eta * Vector.dot(rjk,
swarm.vehicle[vid].x)* Vector.dot(rjk, swarm.vehicle[vid].y) - alpha    * (1
- (ro / norm) * (ro / norm))* Vector.dot(rjk, swarm.vehicle[vid].y)+
Vector.dot(tempkx, swarm.vehicle[vid].y);

                        tempu = sumterm + tempu;
                        return tempu;
                  }

            public void pureControl(Swarm swarm){
                  double totu;
                  for(int i = 0; i<Swarm.num; i++){
                        double rectu = calcControlRect(i, swarm);
                        double circu = calcControlCirc(i, swarm);
                        double wptu = calcWptControl(i, swarm);
                        if(wpts.wptList[wpts.place].delay>0 &&
wpts.distToWpt(swarm.vehicle[i].Rxy)<=wpts.dist){
      //Circular Control
                              totu = circu + wptu;
                        }
                        else{
                        // Rectilinear Control
                              totu = rectu + wptu;
                        }
                        swarm.vehicle[i].setU(totu);
                  }
            }

            public void blendControl(double maxd, double mind, Swarm swarm){
                  double totu=0;
                  double trans=0;
                  for(int i = 0; i<Swarm.num; i++){
                        double rectu = calcControlRect(i, swarm);
                        double circu = calcControlCirc(i, swarm);
                        double wptu = calcWptControl(i, swarm);
                        wpts.getPosition();
                        this.rjk.elem[0] = swarm.vehicle[i].Rxy.elem[0]
                  - wpts.wptList[wpts.place].elem[0];
                        this.rjk.elem[1] = swarm.vehicle[i].Rxy.elem[1]
                  - wpts.wptList[wpts.place].elem[1];
                        this.norm = rjk.norm();
                        if(norm>maxd){
                              trans=0;
                        }
                        if(norm<mind){
```

```
                        trans=1;
                    }
                    if(norm<maxd && norm>mind){
                        trans=1-(1/(maxd-mind))*(norm-mind);
                    }
                    if(trans==0){
                        totu= rectu+ wptu;
                    }
                    if(trans==1){
                        totu= circu+ wptu;
                    }
                    if(trans<1 && trans>0){
                        totu= ((1-trans)*rectu+trans*circu)+ wptu;
                    }
                    swarm.vehicle[i].setU(totu);
                }
            }

    public Matrix vehicDist(Swarm swarm){
        double norm = 0;
        Matrix sep = new Matrix((int)swarm.num, (int)swarm.num,
    norm);
        for(int i = 0; i<Swarm.num; i++){
        for (int k = 0; k < swarm.vehicle.length; k++) {
            if (k != i) {
this.rjk.elem[0] = swarm.vehicle[i].Rxy.elem[0] -
swarm.vehicle[k].Rxy.elem[0];

this.rjk.elem[1] = swarm.vehicle[i].Rxy.elem[1] -
swarm.vehicle[k].Rxy.elem[1];

                    this.norm = rjk.norm();
                    sep.elem[i][k]= rjk.norm();
                    }
                }
            }
            return sep;
    }

}
```

3.6. Log.java

```
package advancedControl;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

public class Log {

    // Creates new waypoint object
    Waypoint wpts = new Waypoint();

    // File Variables
    private PrintWriter pw;
```

```
private PrintWriter pwt;
private PrintWriter pwu;
private PrintWriter pwd;

public Log() {
} // Constructor

//Vehicle Positions Log
public void openLog() { //Open Log for Vehicle Position Data
	try {
		File data = new File("data.m");
		FileWriter fw = new FileWriter(data);
		this.pw = new PrintWriter(fw);
	} catch (IOException e) {
		System.out.println(e);
	}
}
//Log vehicle positions x,x,y,y
public void dataLog(double numvehicles, Swarm swarm) {
	for (int j = 0; j <= numvehicles - 1; j++) {
		pw.print(swarm.vehicle[j].Rxy.elem[0]);
		pw.print(",");
	}
	for (int j = 0; j <= numvehicles - 1; j++) {
		pw.print(swarm.vehicle[j].Rxy.elem[1]);
		pw.print(",");
		if (j == numvehicles - 1) {
			pw.println("");
		}
	}
}

public void closeLog() {//Close Log for Vehicle Position Data
	pw.close();
}

//Steering Command u log
public void openULog() {//Open Log for Vehicle Position Data
	try {
		File data = new File("u.m");
		FileWriter fw = new FileWriter(data);
		this.pwu = new PrintWriter(fw);
	} catch (IOException e) {
		System.out.println(e);
	}
}
//Log vehicle positions x,x,y,y
public void ULog(double numvehicles, Swarm swarm) {
	for (int j = 0; j <= numvehicles - 1; j++) {
		pwu.print(swarm.vehicle[j].u);
		pwu.print(",");
		if (j == numvehicles - 1) {
			pwu.println("");
		}
	}
}
```

```
	public void closeULog() {//Close Log for Vehicle Position Data
		pwu.close();
	}
	//Vehicle Separation Distance Log
	public void openDistLog() {   //Open Log for Separation Distance
		try {
			File data = new File("sepdist.m");
			FileWriter fw = new FileWriter(data);
			this.pwd = new PrintWriter(fw);
		} catch (IOException e) {
			System.out.println(e);
		}
	}
	public void logDist(double numvehicles, Matrix rsep){
		for (int j = 0; j <= numvehicles - 1; j++) {
			for(int i = 0; i<= numvehicles-1; i++){
				pwd.print(rsep.elem[j][i]);
				pwd.print(",");
			}
			if (j == numvehicles - 1) {
				pwd.println("");
			}
		}
	}
	public void closedDistLog() { //Close Log for Vehicle Separation
		pwd.close();
	}
	//Waypoint Log
	//Log waypoint positions x,x,y,y
	public void WptLog(double numvehicles, Waypoint wpts) {
		for (int j = 0; j <= numvehicles - 1; j++) {
			pwt.print(wpts.wptList[j].elem[0]); // X positions
			pwt.print(",");
		}
		for (int j = 0; j <= numvehicles - 1; j++) {
			pwt.print(wpts.wptList[j].elem[1]); // Y positions
			pwt.print(",");
			if (j == numvehicles - 1) {
				pwt.println("");
			}
		}
	}
	public void openWptLog() {          //Open log for waypoint data
		try {
			File data = new File("wpts.m");
			FileWriter fw = new FileWriter(data);
			this.pwt = new PrintWriter(fw);
		} catch (IOException e) {
			System.out.println(e);
		}
	}

	public void closeWptLog() {         //Close log for waypoint data
		pwt.close();
	}
}
```

3.7. Simulate.java

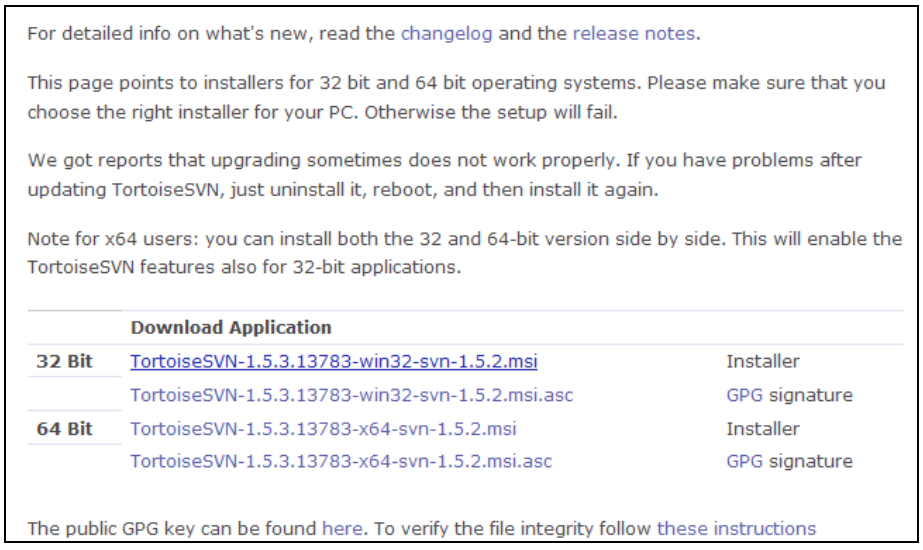
```
package advancedControl;

public class Simulate {

	public static double dt;
	public static double tfin;
	public static double t;
	public static int n;
	public static double maxd;
	public static double mind;
	public static double count;

	public static void main(String[] args) {

		// Simulation Paramaters
		dt = 0.1;
		tfin = 2000;
		n = 6;
		maxd = 300;
		mind = 50;
		//Create objects for simulation
		Swarm myswarm = new Swarm();
		Waypoint waypoints = new Waypoint();
		Log data = new Log();
		Control control= new Control();
		data.openLog();    //Open vehicle position log
		data.openULog();   //Open steering command log
		data.openDistLog();//Open vehicle separation distance log
		myswarm.initSwarm();          //Initialize swarm

		for( double t=0; t<=tfin; t=t+dt){
			//Blended Control
			//control.blendControl(maxd, mind, myswarm);

			//Pure Rectilinear or Circular control
			control.pureControl(myswarm);

			//Move swarm and update position
			myswarm.moveSwarm();
			data.dataLog(n, myswarm);     //Log vehicle positions
			data.ULog(n, myswarm);        //Log steering command

			//Logs vehicle separation distance matrix
			data.logDist(n, control.vehicDist(myswarm));
		}
		data.closeLog();        //Close vehicle position log
		data.closeULog();       //Close steering command log
		data.closedDistLog();   //Close vehicle separation log
		// Waypoint Position File
		data.openWptLog();
		data.WptLog(n, waypoints);
		data.closeWptLog();
	}
}
```

4. Naval Research Laboratory Multi-Vehicle Simulator Installation Procedure

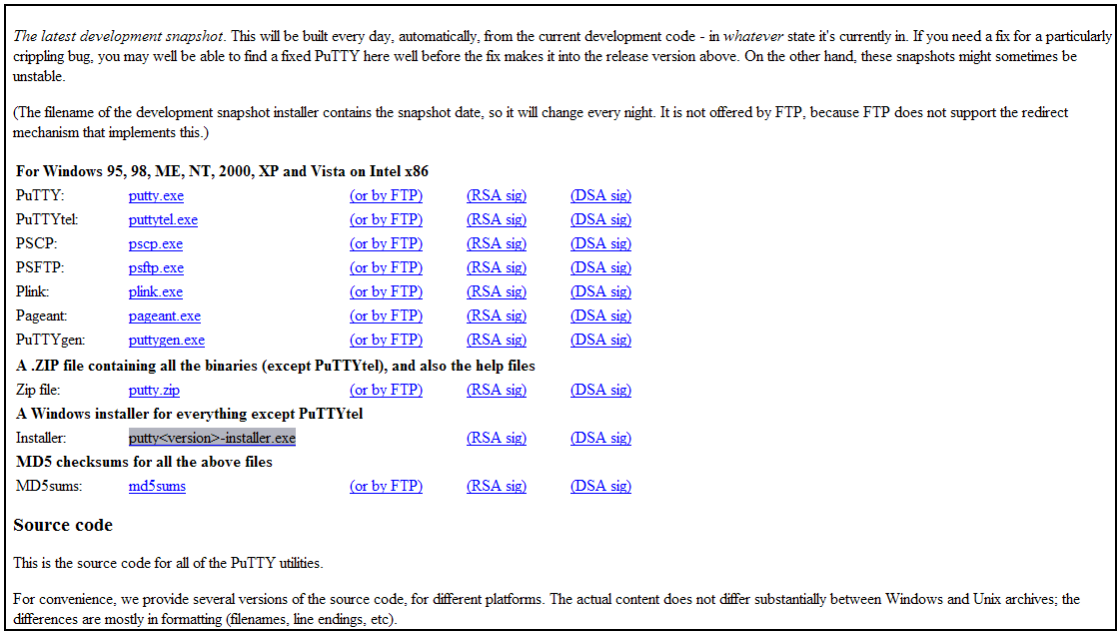
1. **Java SE Development Kit (SDK)**
   a. http://java.sun.com/javase/downloads/index.jsp
   b. Select Platform: Windows, Language: Multi Language, Continue
   c. Under required files, click Windows Offline Installation, Download

2. **Eclipse Classic**
   a. http://www.eclipse.org/downloads/
   b. Click on Eclipse Classic 3.4 (or latest version)
   c. Chose a mirror close to current location
   d. Open with WinZip
   e. Click on eclipse.exe
   f. Set up a workspace location to store all your projects

3. **TortiseSVN**
   a. http://tortoisesvn.net/downloads
   b. Download the 32bit Installer
   c. Next -> Install -> Finish -> Restart



4. **PuTTY**
    a. http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
    b. Latest Development Snapshot -> Windows Installer
    c. Install



The latest development snapshot. This will be built every day, automatically, from the current development code - in *whatever* state it's currently in. If you need a fix for a particularly crippling bug, you may well be able to find a fixed PuTTY here well before the fix makes it into the release version above. On the other hand, these snapshots might sometimes be unstable.

(The filename of the development snapshot installer contains the snapshot date, so it will change every night. It is not offered by FTP, because FTP does not support the redirect mechanism that implements this.)

For Windows 95, 98, ME, NT, 2000, XP and Vista on Intel x86

PuTTY:	putty.exe	(or by FTP)	(RSA sig)	(DSA sig)
PuTTYtel:	puttytel.exe	(or by FTP)	(RSA sig)	(DSA sig)
PSCP:	pscp.exe	(or by FTP)	(RSA sig)	(DSA sig)
PSFTP:	psftp.exe	(or by FTP)	(RSA sig)	(DSA sig)
Plink:	plink.exe	(or by FTP)	(RSA sig)	(DSA sig)
Pageant:	pageant.exe	(or by FTP)	(RSA sig)	(DSA sig)
PuTTYgen:	puttygen.exe	(or by FTP)	(RSA sig)	(DSA sig)

A .ZIP file containing all the binaries (except PuTTYtel), and also the help files

Zip file:	putty.zip	(or by FTP)	(RSA sig)	(DSA sig)

A Windows installer for everything except PuTTYtel

Installer:	putty<version>-installer.exe		(RSA sig)	(DSA sig)

MD5 checksums for all the above files

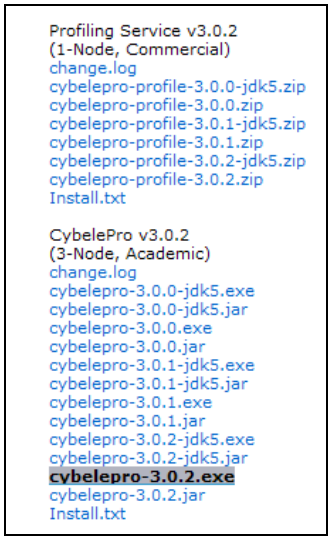
MD5sums:	md5sums	(or by FTP)	(RSA sig)	(DSA sig)

Source code

This is the source code for all of the PuTTY utilities.

For convenience, we provide several versions of the source code, for different platforms. The actual content does not differ substantially between Windows and Unix archives; the differences are mostly in formatting (filenames, line endings, etc).

5. **CybelePro**
    a. www.cybelepro.com/login/login.asp
    b. Login using username and password
    c. Download liscense.prop file (not on website)
    d. Download latest version academic



Profiling Service v3.0.2
(1-Node, Commercial)
change.log
cybelepro-profile-3.0.0-jdk5.zip
cybelepro-profile-3.0.0.zip
cybelepro-profile-3.0.1-jdk5.zip
cybelepro-profile-3.0.1.zip
cybelepro-profile-3.0.2-jdk5.zip
cybelepro-profile-3.0.2.zip
Install.txt

CybelePro v3.0.2
(3-Node, Academic)
change.log
cybelepro-3.0.0-jdk5.exe
cybelepro-3.0.0-jdk5.jar
cybelepro-3.0.0.exe
cybelepro-3.0.0.jar
cybelepro-3.0.1-jdk5.exe
cybelepro-3.0.1-jdk5.jar
cybelepro-3.0.1.exe
cybelepro-3.0.1.jar
cybelepro-3.0.2-jdk5.exe
cybelepro-3.0.2-jdk5.jar
cybelepro-3.0.2.exe
cybelepro-3.0.2.jar
Install.txt

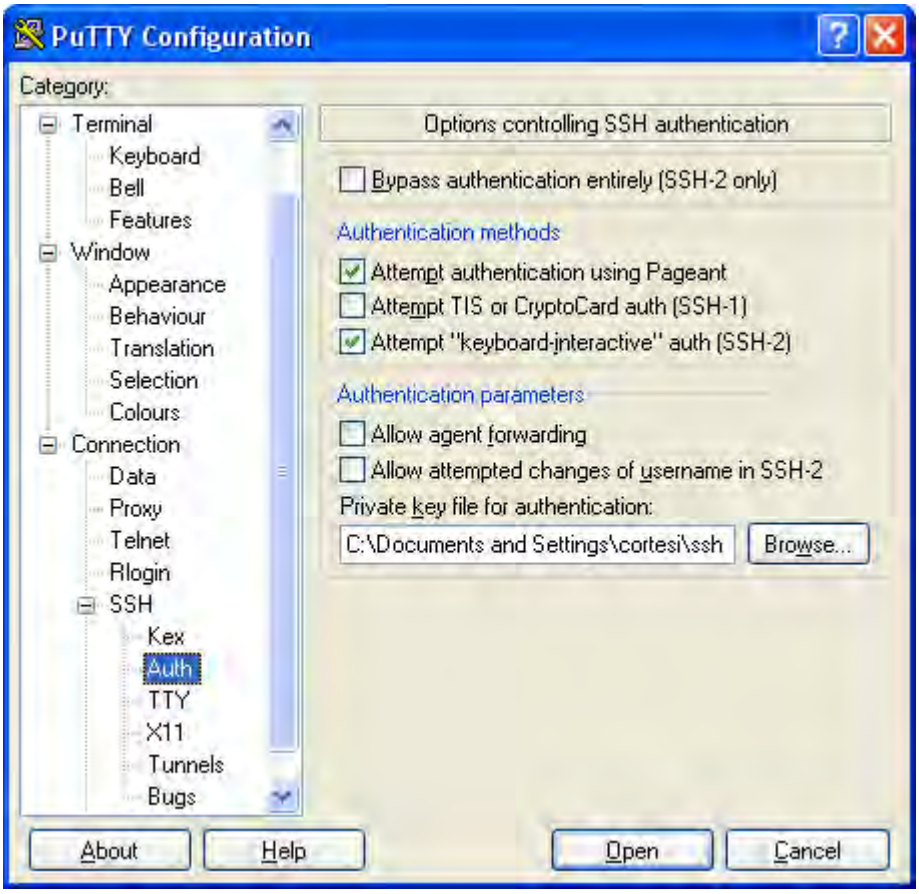
6. Optional – Pin Putty, Pageant, TortoiseSVN, Eclipse to start menu
    a. Start -> Programs -> Desired Program ->Right Click ->Pin to Start Menu
7. Obtain SSH key and pass phrase and store on relevant computers
8. **Configure PuTTY**

a. Add the path to the PuTTY executable directory to the path environment variable.
    i. Command prompt type “set PATH=C:\Program Files\ PuTTY
b. Test this by running PuTTY from the command line. If the path was added successfully, then you should just be able to type “putty” at the command prompt (i.e. the full path is not required).
c. Copy your private key to the mvsim repository to the local machine. I recommend creating a directory called “ssh”.
d. Set up Pageant to take care of authentication by following these steps. First, if you don't see the Pageant icon (a computer wearing a hat) in the system tray, you'll need to start Pageant by going to Start - All Programs - PuTTY - Pageant. Then right-click on the icon in the system tray and choose "add key". Browse to the location where you saved your private key file. Then Pageant will ask you to enter your passphrase associated with that key. Once you've entered this, Pageant will take care of all authentication for the rest of that session (i.e. if you log out or restart, you'll have to do this step again).
e. **Tunneling**
   If you are connecting to the repository from off site you must use ssh (or PuTTY) to tunnel in, to connect to the mvsim computer. You must have an active connection to **spg.nrl.navy.mil** to read or write from the simulator repository. Open Putty and enter the following:

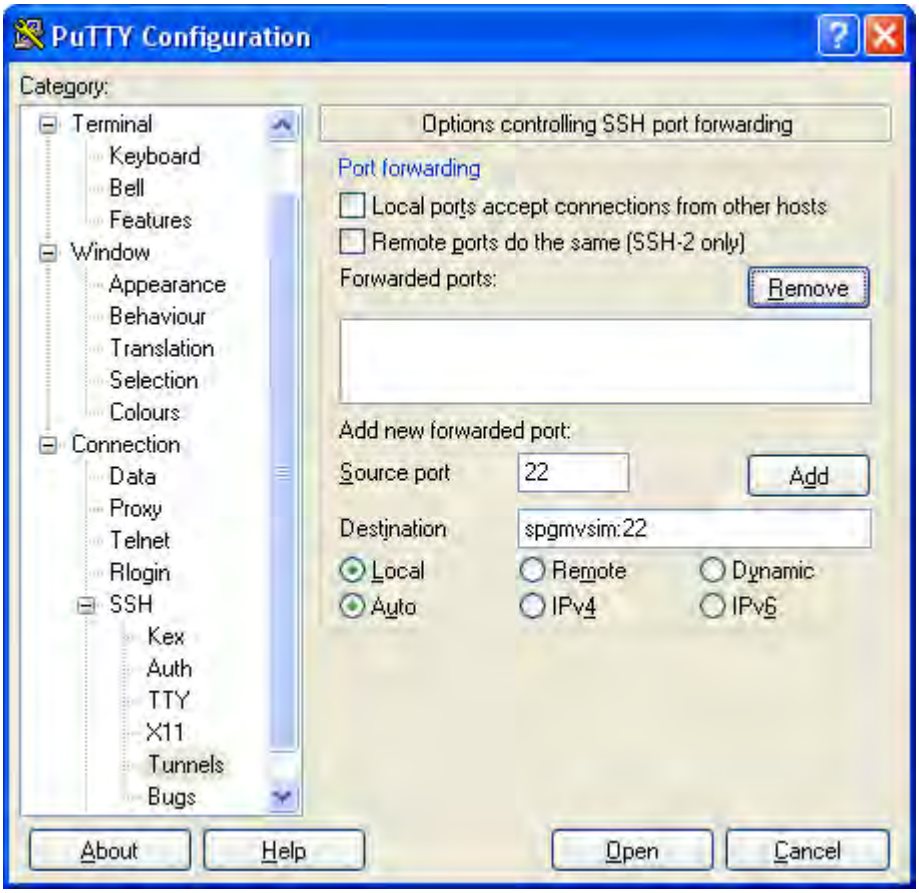
Category	Field	Value
Session	Host Name (or IP address)	username@spg.nrl.navy.mil
Session	Port	22
Session	Saved Session	<any name you want>
Connection → ssh → Auth	Private key file for authentication	<browse and select your private key file>
Connection → ssh → Tunnels	Source port	22
Connection → ssh → Tunnels	Destination	spgmvsim:22 <then press add>

Go back to the session screen in PuTTY and save the configuration.

Step 1: configure for connecting to spg.nrl.navy.mil on port 22



Step 2: Configure for private key authentication using your private key file.



Step 3: Configure for tunneling to spgmvsim. Then press "Add"

Step 4: Save the configuration by going back to the screen in step 1 and pressing "Save".

9. Configure Tortoise SVN (also see note below about optional install of Subclipse)

a. Go to the TortoiseSVN settings dialog box (right click and choose "settings" in the TortoiseSVN menu).

b. In the general setting enter the following in the "Global ignore pattern:" field "***.class .project .classpath *.metadata Thumbs.db *.log *.aux *.dvi .* *~**"

c. In the network settings enter the following in the "SSH client:" field "tortoiseplink –l <username>"

10. Check out a working copy from the repository:

a. Note: if you are using a roaming profile on the ENEWS domain (I.e. Rob Lacefield set you up so that your Desktop, Preferences, and Documents get imported to wherever you log-in on an ENEWS computer) then you probably don't want to save a working copy of the repository code on your Desktop or in your Documents folder because it will take a long time for the system to log in

every time (the working copy is about 1.5GB and that will have to be transferred over the network every time you log in.) Instead, you probably want to check out the working copy to your space on the "Enewsfs" share drive, which will always be accessible to you but won't have to be transferred across the network every time you log in. However, you must ensure that you check "Save my password" when you first browse to your space on the Enewsfs share drive; otherwise Subversion won't be able to write to the folder and SVN updates will fail.

b. Right-click at the location where you want to save your working copy and go to "SVN Checkout". In the URL box, you'll either type:
    i. If your computer is **located at NRL and on the ENEWS domain**, type "*svn+ssh://spgmvsim/mvsim*".
    ii. If your computer is **located at NRL and NOT on the ENEWS domain**, type "*svn+ssh://spgmvsim.enews.nrl.navy.mil/mvsim*"
    iii. If your computer is **NOT located at NRL** type "*svn+shh://localhost/mvsim*", and see the section on Tunneling in the Configuring PuTTY section..
c. Click OK; Tortoise should start adding the requested files to your working directory. This may take a few minutes.

11. In Eclipse, create a new JAVA project:

a. File → New → Project…
b. Select "Java Project"
c. Select "create project from existing source"
d. Browse to "mvsim/code/java2/mil.navy.nrl.spg" and select OK
e. Name the project "mil.navy.nrl.spg"
f. Select "Next…"
g. Select the "Libraries" tab and click "Add External JARs"
h. Browse to the CybelePro install location. Usually: "C:\Program Files\cybelepro\lib"
i. Select all and click "Open"
j. Click "Finish"

Note: Before running an application using the CybelePro framework. The Cybele-daemon must be running. If the daemon is not running then the Cybele start up will hang at "Contacting IAIDaemon …"

k. In Eclipse, if there are any files with errors that are not necessary
    i. Right click -> Source path -> Exclude
l. Running a basic simulation in eclipse

i. Locate the class with the main function you wish to run (for example MultiBoatSimulation)
ii. Open and edit the simulation config.xml file and the boat initialization.xml file
iii. Open and edit the track.csv file for the convoy waypoints
iv. Run the main class
v. Using the GUI, a typical simulation will run as:
    1. "i" to initialize agents
    2. "s" to begin simulation
vi. Output in the form of a .asi file

12. (Optional) Install Subclipse plug-in

a. Subclipse is a plug-in for the Eclipse IDE which allows you to interact with the Subversion repository through the Eclipse interface. This is convenient because you can update, commit, move, and delete versioned files without having to bounce back and forth between Eclipse and a file explorer window. Also, it prevents you from confusing Subversion by moving/deleting a versioned file in the Eclipse package explorer (I.e. without doing the proper SVN move or SVN delete).

b. Download the Subclipse plug-in (and necessary additional items that are packaged with it) by clicking on "Help" in Eclipse and then selecting Software Updates. On the Available Software tab, click on Add Site on the right hand side. For the URL, put in http://subclipse.tigris.org/update_1.4.x. Click on the box next to "Subclipse" and then click on Install. When it's complete, it will have you restart Eclipse.

c. Find the Subversion configuration file *C:\Documents and Settings\<username>\Application Data\Subversion* called "config".
    i. We need to modify this so that Subclipse will know how to use SSH. Scroll down to the section entitled "tunnels". Uncomment the "ssh" line and replace it with: **`ssh = C:\\Program Files\\TortoiseSVN\\bin\\TortoisePlink.exe -l`** `<put_your_username_here>` (For some reason you have to put in the double slashes).
    ii. We also need to tell subclipse which files to ignore for versioning. Uncomment the "global-ignores" line and make sure it reads as follows, you will have to add some items to the list.
    "global-ignores = *.class Thumbs.db *.log *.aux *.dvi *.o *.lo *.la .*.rej *.rej .*~ *~ .* .DS_Store"

d. Now, in Eclipse when you right click on a file you will find some new options. The main one is a menu item called “Team” which has many of the typical Subversion commands. Also, under Refactor you can do moves and deletes. (Note: The black asterisks are equivalent to the red exclamation marks in TortoiseSVN.)

13. SIMDIS

a. https://simdis.nrl.navy.mil/

b. Create an account and log in

c. Download latest version of SIMDIS

d. Register System to receive license number

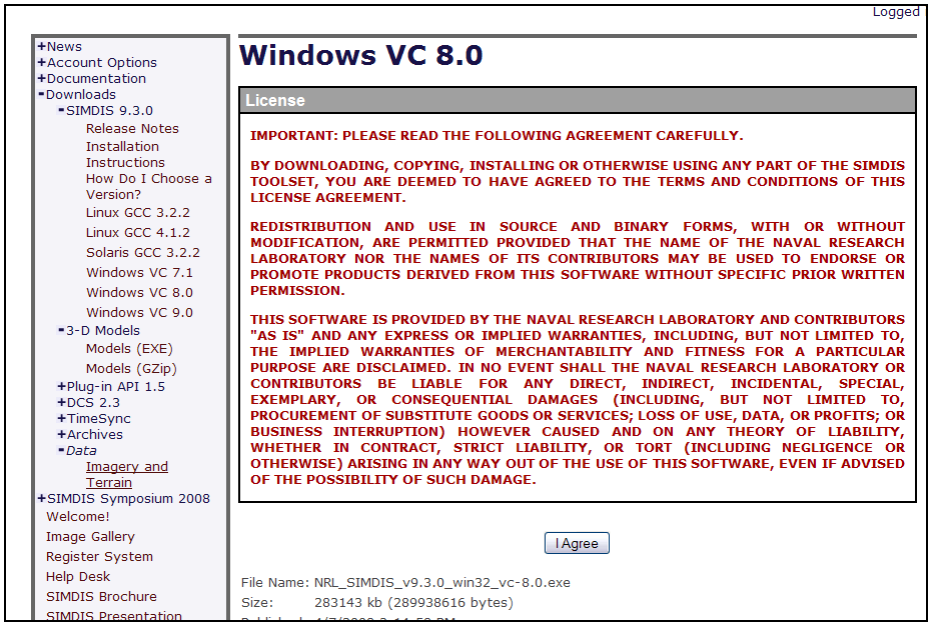
e. To download extra terrain

i. Downloads -> Data -> Imagery and Terrain

ii. Download config text file

iii. Download all .db files stated in the config file

iv. Place .db files in corresponding folders in the SIMDIS folder tree

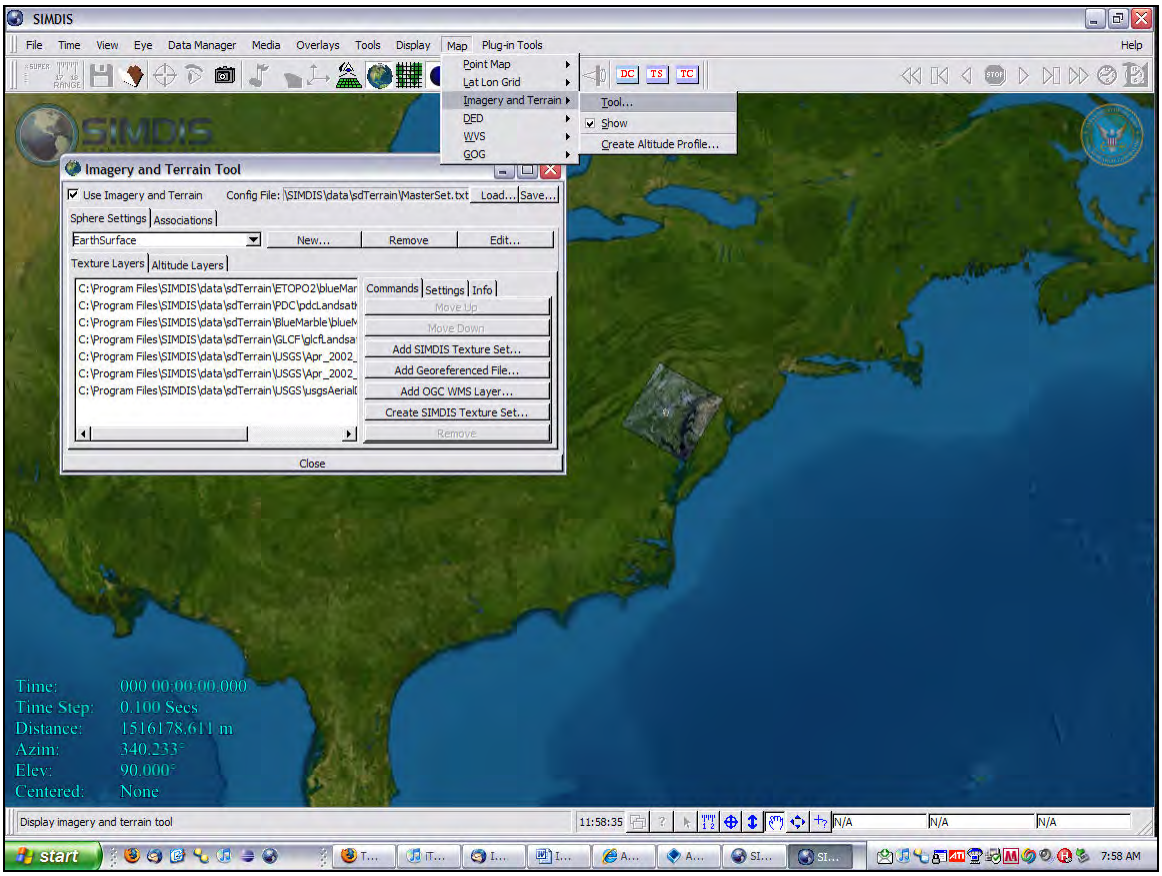


Location of latest version and link for Terrain downloads

v. In SIMDIS Map ->Imagery and Terrain-> Tool-> Load config file

vi. Also possible to make your own config file

1. Load all terrain and altitude files you want and hit save and name the config file



f. To run a simulation

i. Open the .asi file that is outputted by mvsim

14. Garmin GPSmap 60Cx

a. Download and install USB driver

i. http://www8.garmin.com/support/download_details.jsp?id=591

b. Update software version

i. http://www8.garmin.com/support/download_details.jsp?id=1225

c. Install Tracking and Route saving

i. http://www.gpsbabel.org/download.html

ii. Extract the ZIP file and run GPSBabelGUI.exe

1. In input, select Garmin serial/USB protocol
2. In output, select Universal csv with field structure in first line

15. Apache Logger

a. Download apache-log4j-1.2.15.zip

i. http://logging.apache.org/log4j/1.2/download.html

b. Update referenced .jar library in Eclipse
    i. In Eclipse, right click referenced libraries->add external JARs and find the log4j-1.2.15.jar file

16. Automated simulation trials

a. Controls.JusthCalcParamFactory
    i. Modify indexfilename and paramfilename with computers classpath
        1. use "\\" vice "\" for folder separation
b. Add log4j-1.2.15.jar in Cybele/lib folder with Cybele .jar files
    i. In Eclipse, right click referenced libraries->add external JARs and find the log4j-1.2.15.jar file
c. Place log4j.properties in mil.navy.nrl.spg folder
d. batchrun.bat
    i. add classpath in quotes "C:\....\mil.navy.nrl.spg"
    ii. ad number of trials in the for loop
        1. for %%i in (1…..20)
e. JusthCalcParamfactory.properties
    i. make index =1 (or line number of the params.csv file you wish to read)
f. matlabAgent
    i. change output file name if needed
g. params.csv file needed
h. sim_config.xml
    i. make sure track file is correct for convoy
i. smallboat_init.xml
    i. update all information for the vehicle agents
j. To run, double-click batchrun.bat in the source folder vice eclipse

5. *SIMDIS User's Guide*

This guide covers the following topics

- ***SIMDIS initialization***
- ***Creating presentation videos***
- ***Plot desired trajectories***
- ***Sensor Projection***

1. Use SIMDIS_Logger.m to convert data file to .asi file format
    a. Both the Scenario Initialization and the Platform initialization are required
        i. Scenario Initialization must contain a Version name, Ref LLA, and Reference Coordinate System
        ii. Platform Initialization must contain PlatformID, PlatformName and PlatformIcon
        iii. Optional keywords are explained on in the SIMDIS manual Appendix A
    b. Editing input and output filenames

```
%% Initialize variable

fidw = fopen('S1_IP1_119.asi','w');  %.asi file

%% Open data file
 filename = 'Scenario1_119_IP1.txt';
 data = load (filename);
 lla = '0 0 0'; %Reference location (Severn River)
```

Change output filename

Change input filename

Change reference Latitude, Longitude and Altitude (LLA). This allows the user to set a reference position on the earth which all of the data will be based from. This is useful if the input data is not in a latitude, longitude, altitude format but in an x ,y, z format

c. Editing .asi file header information

i. Additional information can be removed/added at user's preference.

```
%% Print Header
fprintf(fidw,'# SIMDIS Ascii Scenario Input (ASI) File Format\n');
fprintf(fidw,'#Created from data in file %s',filename);
fprintf(fidw,'\n');

fprintf(fidw,'Version 16\n');
fprintf(fidw,'ScenarioInfo "UAV trajectory"\n');  %Describe scenario here
fprintf(fidw,'\n');

fprintf(fidw,'RefLLA %s\n',lla);
fprintf(fidw,'CoordSystem "LLA"\n');
fprintf(fidw,'ReferenceYear 1970\n');
fprintf(fidw,'TangentPlaneOffset 0. 0. 0.\n');
fprintf(fidw,'\n');

fprintf(fidw,'Classification "UNCLASSIFIED" 0x8000ff00\n');
fprintf(fidw,'ReferenceTimeECI "0."\n');
fprintf(fidw,'\n');

fprintf(fidw,'WindAngle 0.\n');
fprintf(fidw,'WindSpeed 0.\n');
fprintf(fidw,'\n');

fprintf(fidw,'ITConfigFile "configUSNA.txt"');  %Loads terrain config file
fprintf(fidw,'\n');

fprintf(fidw,'VerticalDatum "WGS84"\n');
fprintf(fidw,'\n');

fprintf(fidw,'PlatformID 1\n');
fprintf(fidw,'PlatformIcon 1 "CRRC"\n');
fprintf(fidw,'PlatformName 1 "AUV"\n');
fprintf(fidw,'PlatformCoordSystem 1 "ENU"\n');        %x-East y-North z-altitude
fprintf(fidw,'CategoryData 1 -1 "Platform Type" "ship"\n');
fprintf(fidw,'\n');
```

Edit Reference LLA and the coordinate system the reference point is given in

Edit default terrain file

Initialize Vehicles

ii. The addition of another vehicle would result in the following additional code:

```
%Convoy
fprintf(fidw,'PlatformID 2\n');
fprintf(fidw,'PlatformIcon 2 "jeep_wrangler"\n');
fprintf(fidw,'PlatformName 2 "Convoy"\n');
fprintf(fidw,'PlatformCoordSystem 2 "LLA"\n');
fprintf(fidw,'CategoryData 2 -1 "Platform Type" "Unknown"\n');
fprintf(fidw,'\n');
```

iii. Depending on the format of the input data, PlatformCoordSystem may need to be changed from LLA to ENU or others.

d. Data input for platform

```
%% Print logged data
% Format Used (pg. 367)
%  Time, position, orientation and speed

for(i=1:size(data,1))
fprintf(fidw,'PlatformData 1 %d %d %d %d %d %d 0.0 %d\n', i, data(i,8), data(i,9), -data(i,5), data(i,1) , data(i,12), data(i,3));
end
```

i. PlatformData string must contain the following

1. Platform unique id
2. Time (set to -1 for static entities)
3. X position (meters, radians, degrees)
4. Y position (meters, radians, degrees)
5. Z position (meters)

ii. Optional

1. Yaw or Psi orientation (radians or degrees)
2. Pitch or Theta orientation (radians or degrees)
3. Roll or Phi orientation (radians or degrees)
4. Quantenion scalar and vector (q0,q1,q2,q3) if PlatofrmUsesQuanternion is set in header
5. Speed or Velocity vector (Vx, Vy, Vz) (meters/sec)

iii. The following string formats are permitted:

```
// Minimum input, time and position
PlatformData 1  0.0 1115.0 342.0 0.0
// Time, position and orientation
PlatformData 2  0.0 1115.0 342.0 0.0 3.344 0.0 0.0
// Time, position, orientation and speed
PlatformData 3  0.0 1115.0 342.0 0.0 3.344 0.0 0.0 10.0
// Time, position, orientation and velocity vector
PlatformData 4  0.0 1115.0 342.0 0.0 3.344 0.0 0.0 3.0 5.0 2.0
```

2. Recording video of simulation using Fraps
    a. http://www.fraps.com/download.php
    b. Configure

    c. To capture video
        i. Make sure Fraps is running and minimized
        ii. When you want to begin recording, hit F9 (or assigned video hotkey)
        iii. Press F9 (or assigned video hotkey) to end recording
            1. Output file is .avi
    d. Use video editing software for compression and editing

3. Plot desired trajectories
    a. Initialize as an object for plotting waypoints
        i. Create unique PlatformID, PLatformName and PlatformIcon as any normal vehicle.
        ii. Data input for platform
            1. Platform Unique ID
            2. Time is -1
            3. x, y, z position
        iii. This method will treat waypoints as simulation objects and cause user to have to cycle through all waypoints when trying to center on a vehicle.
    b. Using .gog file
        i. In Platform initialization use the line
            1. PlatformAttachedGOG ‘platform_ID’ “gogfile.gog”
        ii. In .gog file

```
start
linecolor blue
points
ll 38.9871949 -76.485415 0.0
ll 38.9873490 -76.4852175 0.0
ll 38.9875232 -76.4849973 0.0
ll 38.9877459 -76.4847090 0.0
ll 38.987895 -76.4845174 0.0
end
```

        iii. ll command uses lat and long as arguments,
        iv. xy command uses yards as distance arguments
        v. ref command can be used to set a reference point (LLA), then xy used from this reference point
4. Sensor Projection

a. Use beams

i. Initialize beam

1. Must contain BeamID, VertBW (in degrees), HorzBW (in degrees)

```
if(0)
%Beam Initialization
fprintf(fidw,'BeamID 2 201\n');
fprintf(fidw,'VertBW 201 176.64\n'
fprintf(fidw,'HorzBW 201 176.64\n'
fprintf(fidw,'BeamDesc 201 "EM Jam
fprintf(fidw,'\n');
end
```

PlatformID beam attached to then Beam UniqueID

ii. Data input for beam

1. Must contain BeamOnOffCmd and BeamData

```
%Print Beam Data
if(0)
fprintf(fidw,'BeamOnOffCmd %d %d 1\n', 201, data(2,2)); %Turn Beam on at start of simulation data
fprintf(fidw,'BeamData %d %d %s %d %d %d\n', 201, data(2,2), 'green', 0.0, 1.54, 100);
end
```

2. BeamOnOffCmd must contain
    a. Beam unique ID
    b. Time to start
    c. Beam state (0:off, 1:on)
3. BeamData must contain
    a. Beam unique ID
    b. Time to start
    c. Color (string or hex number)
4. Optional for BeamData
    a. Azimuth (rad or degrees, CW from N)
    b. Elevation (rad of degrees, + above horizon)

c. Range (meters from host platform)

d. Can also track an assigned “target”

i. See Appendix A for more details

b. Use .gog file

i. In Platform initialization use the line

1. PlatformAttachedGOG ‘platform_ID’ “gogfile.gog”

ii. In .gog file

```
start
circle
linecolor red
3d follow c
center xy 0 0 -100
radius 1760
end
```

iii. ‘3d follow c’ makes the gog object follow the attached platform’s course

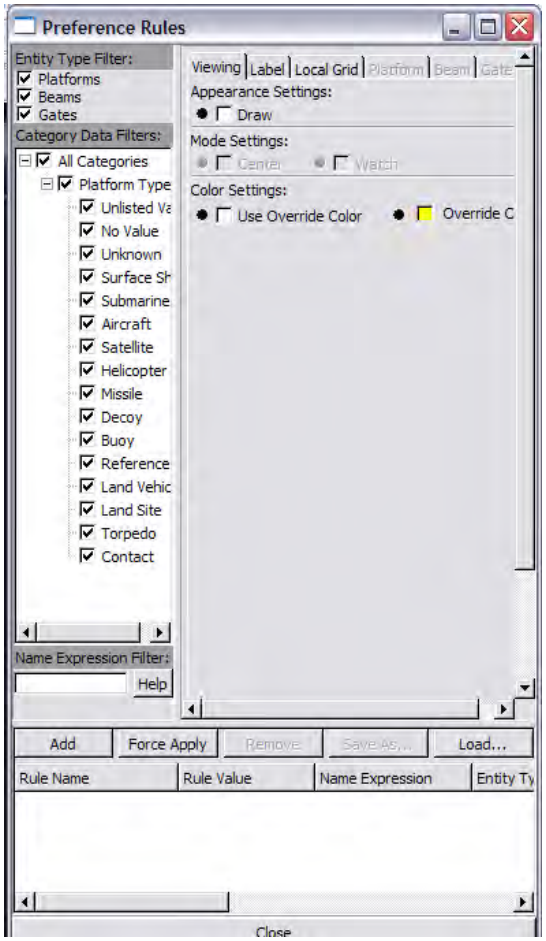
iv. ‘center xy’ – x and y values in feet, z value in yards

v. ‘radius’ value in yards

5. Edit Preference Rules

a. Used to edit the display properties of the Platforms

b. Tools-> Preference Rules



c. Save updated preference file

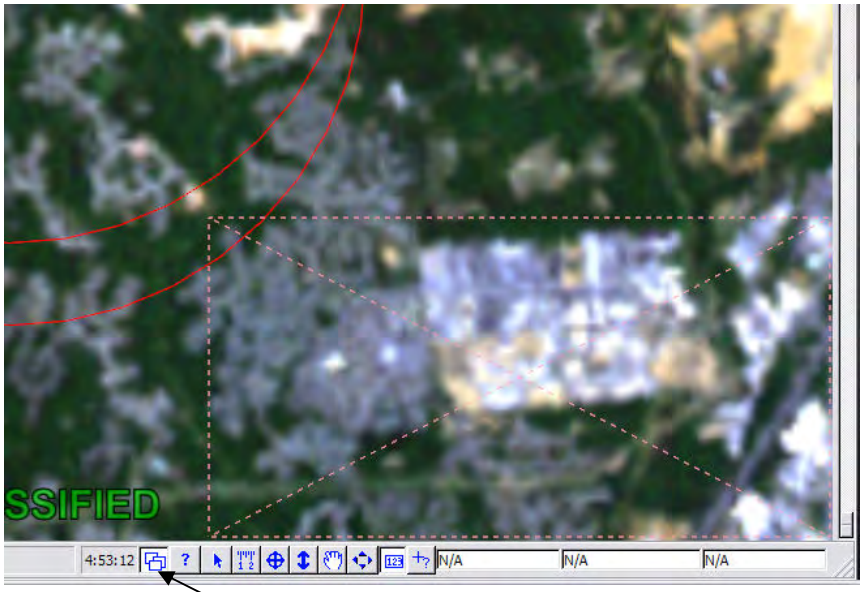
d. Load in .asi file using RuleFile keyword

   i. RuleFile “my_rules.rul”

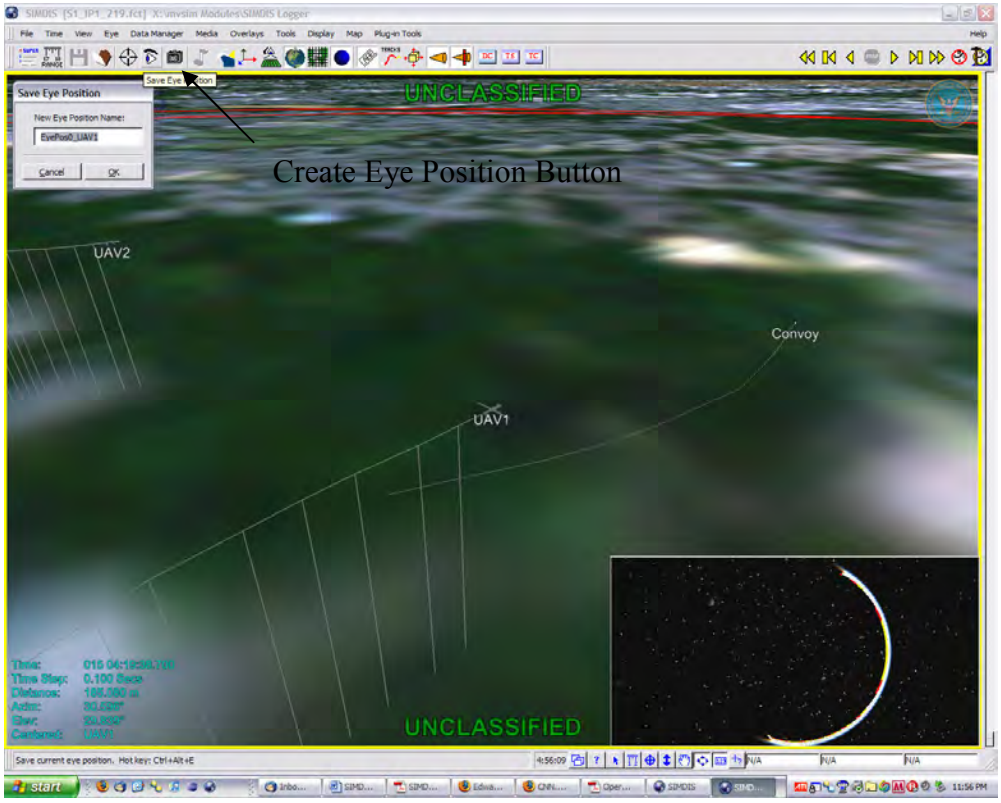
6. Edit Views

   a. Create a viewport (picture in picture)

      i. Click the viewport button and draw a box that will be your future viewport

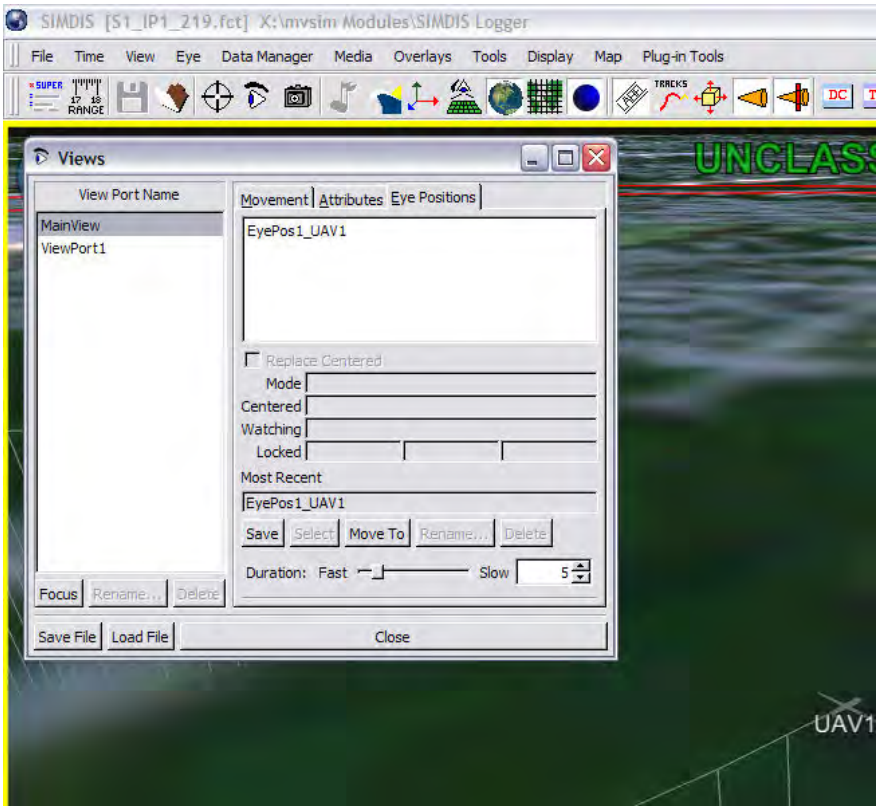


b. Save eye positions



c. Save view

i. View -> Views



ii. View Port Name shows all view ports cretated, and current active view port

iii. Movement, Attributes, Eye Positions options covered in 7.3.1 in the SIMDIS manual

iv. Mainly, select the viewport on the left, the eye position on the right, save and then save the view file with the “Save File” button

6. *Data Analysis Scripts*

6.1. matlabScript_V2.m

```
%%=====================================================================
% File Name    : matlabScript_V2.m
% Authors      : W. Selby
% Date         : 08-Sep-2008
% Description  : Plots output of Java .m/.txt file, stores data in text
% file
%
% Inputs:      : data text file
% Outputs:     : plots
% Requirements : mvsimout.txt, plot output directory at end
% Revisions    : Improved Cost function - linearized convoy penalization,
% computed avergae steering commands
%=====================================================================

%% Initialize
clear; close all; format compact; clc; format long;

%Print Header of .csv file
fidw = fopen('Scenario_data_output_file.csv','w');
fprintf(fidw,'%s,%s,%s,%s,%s,%s,%s,%s\n','Trial','Mu','Alpha','Eta','Ro','%Ou
t of Range','%Too Close','%Max u');

%% Load Files
for(f=1:82)
    filename = sprintf('scenario3_%d_simulation_data.txt',f);
    data = load (filename);

%% Reassign Data to variables

    %File directory
    dir = cd;
    dir_plot = sprintf('%s\\plots',dir);

    % Time
    base_time = data(2,2);

    % Number of Vehicles
    n = 2;

    %Distance Tolerances
    r_min = .031068; %miles = 50 meters
    r_max = 1;    %mile

    %Paramters
    alpha = data(1,2);
    mu = data(1,1);
    eta = data(1,3);
    ro = data(1,4);

    %Initialize variables
```

```
    i=1;w=1;x=0;y=1;z=1;violatec=0;violatev=0;totalc=0;totalv=0;
    percentagec=0;percentagev=0;V1=0;violatectot=0;
    u_tot=0; u_max=60*pi/180; % corresponds to 60deg max rudder angle
    count=1;

%% Vehicle Positions
    for(j=2:size(data,1))
        if(data(j,1)== 4)
            x = x+1;
            V1(x,1) = data(j,2)-base_time; %elapsed time
            V1(x,2) = data(j,3); %lat
            V1(x,3) = data(j,4); %lon
            a(x,1) = sin((V1(x,2) - V1(x,2))/2)^2 +
cos(V1(x,2))*cos(V1(x,2))*sin((V1(x,3) - V1(x,3))/2)^2;
        end
        if(data(j,1)== 2)
            V2(y,1) = data(j,2)-base_time; %elapsed time
            V2(y,2) = data(j,3); %lat
            V2(y,3) = data(j,4); %lon

            if(V1 ~= 0)
                a(y,2) = sin((V2(y,2) - V1(x,2))/2)^2 +
cos(V1(x,2))*cos(V2(y,2))*sin((V2(y,3) - V1(x,3))/2)^2;
                y = y+1;
            end
        end
        if(data(j,1)== 3)
            V3(z,1) = data(j,2)-base_time; %elapsed time
            V3(z,2) = data(j,3); %lat
            V3(z,3) = data(j,4); %lon
            if(V1 ~= 0)
                a(z,3) = sin((V3(z,2) - V1(x,2))/2)^2 +
cos(V1(x,2))*cos(V3(z,2))*sin((V3(z,3) - V1(x,3))/2)^2;
                z = z+1;
            end
        end
        if(data(j,1)== 1)
            V4(w,1) = data(j,2)-base_time; %elapsed time
            V4(w,2) = data(j,3); %lat
            V4(w,3) = data(j,4); %lon
            if(V1 ~= 0)
                a(w,4) = sin((V4(w,2) - V1(x,2))/2)^2 +
cos(V1(x,2))*cos(V4(w,2))*sin((V4(w,3) - V1(x,3))/2)^2;
                w = w+1;
            end
        end
        j = j+1;
    end

%% Calculate average steering command u from vehicle positions

%V2(i,1) =elapsed time
%V2(i,2) =lattitude
%V2(i,3) =longitude
%V1 is the convoy vehicle
```

```
%Calculate u per time step for UAV 1
for(i=10:10:size(V2,1)-30) %one calc per second

    vector1_2a = sin((V2(i+10,2) - V2(i,2))/2)^2 +
cos(V2(i,2))*cos(V2(i+10,2))*sin((V2(i+10,3) - V2(i,3))/2)^2;
    vector1_2c = 2*atan2(sqrt(vector1_2a),sqrt(1-vector1_2a));
    a_u = 3956*vector1_2c*1609.344;

    vector2_3a = sin((V2(i+20,2) - V2(i+10,2))/2)^2 +
cos(V2(i+10,2))*cos(V2(i+20,2))*sin((V2(i+20,3) - V2(i+10,3))/2)^2;
    vector2_3c = 2*atan2(sqrt(vector2_3a),sqrt(1-vector2_3a));
    b_u = 3956*vector2_3c*1609.344;

    vector1_3a = sin((V2(i+20,2) - V2(i,2))/2)^2 +
cos(V2(i,2))*cos(V2(i+20,2))*sin((V2(i+20,3) - V2(i,3))/2)^2;
    vector1_3c = 2*atan2(sqrt(vector1_3a),sqrt(1-vector1_3a));
    c_u = 3956*vector1_3c*1609.344;

    s =(a_u+b_u+c_u)/2;

    if(a_u~=0 && b_u~=0 && c_u~=0 && s-c_u>0)
        k2 = (4*sqrt(s*(s-a_u)*(s-b_u)*(s-c_u)))/(a_u*b_u*c_u);
        u2(i,1)=k2*a_u;
    end
end

%Calculate u per time step for UAV 2
for(i=10:10:size(V3,1)-30) %one calc per second

    vector1_2a = sin((V3(i+10,2) - V3(i,2))/2)^2 +
cos(V3(i,2))*cos(V3(i+10,2))*sin((V3(i+10,3) - V3(i,3))/2)^2;
    vector1_2c = 2*atan2(sqrt(vector1_2a),sqrt(1-vector1_2a));
    a_u = 3956*vector1_2c*1609.344;

    vector2_3a = sin((V3(i+20,2) - V3(i+10,2))/2)^2 +
cos(V3(i+10,2))*cos(V3(i+20,2))*sin((V3(i+20,3) - V3(i+10,3))/2)^2;
    vector2_3c = 2*atan2(sqrt(vector2_3a),sqrt(1-vector2_3a));
    b_u = 3956*vector2_3c*1609.344;

    vector1_3a = sin((V3(i+20,2) - V3(i,2))/2)^2 +
cos(V3(i,2))*cos(V3(i+20,2))*sin((V3(i+20,3) - V3(i,3))/2)^2;
    vector1_3c = 2*atan2(sqrt(vector1_3a),sqrt(1-vector1_3a));
    c_u = 3956*vector1_3c*1609.344;

    s =(a_u+b_u+c_u)/2;

    if(a_u~=0 && b_u~=0 && c_u~=0 && s-c_u>0)
        k3 = (4*sqrt(s*(s-a_u)*(s-b_u)*(s-c_u)))/(a_u*b_u*c_u);
        u3(i,1)=k3*a_u;
    end
end

%Use 2 norm to get average u per time step
for(i=1:min([size(u2,1),size(u3,1)]))
```

```
    if(i<=size(u3,1))
        u_tot(i,1)=100*sqrt(((u2(i,1)/u_max)^2+(u3(i,1)/u_max)^2)/2);
    end
end

%Calculate average steering command
for(i=1:size(u_tot,1))
    if(u_tot(i,1)>0)
        count=count+1;
    end
end
u_avg = sum(u_tot)/count;

%% Log Vehicle Separation Distances
    for(j=1:size(V2,1))
        b(j,1)= sin((V2(j,2) - V3(j,2))/2)^2 +
cos(V3(j,2))*cos(V2(j,2))*sin((V2(j,3) - V3(j,3))/2)^2;%dist from 2-3
        %b(j,2)= sin((V2(j,2) - V4(j,2))/2)^2 +
cos(V4(j,2))*cos(V2(j,2))*sin((V2(j,3) - V4(j,3))/2)^2;%dist from 2-4
        %b(j,3)= sin((V3(j,2) - V4(j,2))/2)^2 +
cos(V4(j,2))*cos(V3(j,2))*sin((V3(j,3) - V4(j,3))/2)^2;%dist from 3-4
    end
    for(j=1:size(V2,1))
        if(n>2)
            for(i=1:n)
                rsep(j,i)= 3956*2*atan2(sqrt(b(j,i)),sqrt(1-b(j,i)));
            end
        else
            rsep(j,1)= 3956*2*atan2(sqrt(b(j,1)),sqrt(1-b(j,1)));
        end
    end

%% Fill Distance to convoy array
    for(i=1:size(V2,1)) %time step
        for(j=1:(n+1))      %vehicle distance%
            c(i,j) = 2*atan2(sqrt(a(i,j)),sqrt(1-a(i,j)));
            d(i,j) = 3956*c(i,j);
        end
    end

%% Data Analysis

    %Plot Distance to Convoy if convoy is vehicle 1
    if(0)
        plotname = sprintf('Scenario#3_Plot_%d_linear_dist_Blend',f);
        h = figure(1);
        subplot(3,1,1);
        ylabel('Distance (miles)');
        hold on; grid on;
        xlim([0 size(V2,1)/10])
        line([0,size(V2,1)],[r_max,r_max],'Color','r');
        for(j=2:(n+1))
            plot(V2(:,1),d(:,j));
            for(i=1:size(V2,1))
                if d(i,j)>r_max
                    violatec = 1;
```

```
                end
                if d(i,j)<r_min
                    violatec = 0;
                end
                if (d(i,j)<r_max && d(i,j)>r_min)
                    violatec = (d(i,j)-r_min)/(r_max-r_min);
                end
                violatectot = violatectot+violatec;
                totalc = totalc+1;
            end
        end
        legend(['Max Distance ', num2str(r_max),' mile(s)'],'UAV 1'...,'UAV
2','UAV 3'
            );
        percentagec = 100*(violatectot/totalc);
        title({['Vehicle Separation Distances - ', num2str(percentagec),'%
range from convoy violation'];[' \alpha = ', num2str(alpha),', \mu = ',
num2str(mu), ', \eta = ', num2str(eta), ', ro = ', num2str(ro)]});

        %Plot Intra-vehicle distance
        subplot(3,1,2)
        ylabel('Distance (miles)');
        hold on; grid on;
        xlim([0 size(V2,1)/10])
        line([0,size(V2,1)],[r_min,r_min],'Color','r');
        for(j=1:(n-1))  %2 vehicles n-1, 3 vehicles n
            plot(V2(:,1),rsep(:,j));
            for(i=1:size(V2,1))
                if rsep(i,j)<r_min
                    violatev = violatev+1;
                end
                totalv = totalv+1;
            end
        end
        legend(['Min Distance ', num2str(r_min),' miles/50 meters'],'UAV 1-
2'...,'UAV 1-3','UAV 2-3'
            );
        percentagev = 100*(violatev/totalv);
        title(['Vehicle Separation Distances - ', num2str(percentagev),'%
intra-vehicle disance violaion']);

        %Plot steering command
        subplot(3,1,3)
        xlabel('Time (seconds)');
        ylabel('Percentage');
        xlim([0 size(V2,1)/10])
        hold on; grid on;
        plot(V2(1:size(u_tot,1),1),u_tot)
        title(['Average Steering Command (u) ', num2str(u_avg),'% of maximum
steering command ']);

        %Save Plot
        cd(dir_plot) %Open plot folder
        saveas(h,plotname);
        cd('..'); %Return to active directory
        close (h);
```

```
    end
%% Write data to .csv file
    %Distance to Convoy Violation
    for(j=2:(n+1))
        for(i=1:size(V2,1))
            if d(i,j)>r_max
                violatec = 1;
            end
            if d(i,j)<r_min
                violatec = 0;
            end
            if (d(i,j)<r_max && d(i,j)>r_min)
                violatec = (d(i,j)-r_min)/(r_max-r_min);
            end
            violatectot = violatectot+violatec;
            totalc = totalc+1;
        end
    end
    percentagec = 100*(violatectot/totalc);

    %Distance between Vehicles Violation
    for(j=1:(n-1))  %2 vehicles n-1, 3 vehicles n
        for(i=1:size(V2,1))
            if rsep(i,j)<r_min
                violatev = violatev+1;
            end
            totalv = totalv+1;
        end
    end
    percentagev = 100*(violatev/totalv);

    %Print Data line to file

fprintf(fidw,'%d,%s,%s,%s,%s,%s,%s,%s\n',f,num2str(mu),num2str(alpha),num2str
(eta),num2str(ro),num2str(percentagec),num2str(percentagev),num2str(u_avg));
end
```

6.2. Compute_Average_v2.m

```
%%=====================================================================
% File Name    : Compute_Average_v2.m
% Authors      : W. Selby
% Date         : 26-Sep-2008
% Description  : Computes cost function
%
% Inputs:      : data text file
% Outputs:     : data text file
% Requirements : ParamDistData per IP
% Revisions    : None
%=====================================================================
%% Initialize
clear; close all; format compact; clc; format short;

%% Declare variables
n = 8; %number of trials
```

```
rho = .25; % performance weighting value

%% Load Files
fidw = fopen('Results_Avg_S1_Route1.csv','w');
fprintf(fidw,'%s,%s,%s,%s,%s,%s,%s,%s,%s\n','Trial','Mu','Alpha','Eta','Ro
(scaled)','Convoy','Collision','E_u','Performance Average');

%% Compute 1 norm for convoy distance and steering control inf-norm for
%% collisions
for(j=1:16)
    filename = sprintf('ParamDistData_v2_%d.csv',j);
    data = load(filename);
    for(i=1:size(data,1))
        if(j==1)
            fx(i+1,5:7)=0;
        end
        gx(i,1) = data(i,6);
        collide(i,j)=data(i,7);
        gx(i,2) = data(i,8);
        fx(i,5) = fx(i,5) + gx(i,1)/n; %final average of convoy distance %'s
        fx(i,6)= max(collide(i,:));
        fx(i,7) = fx(i,7) + gx(i,2)/n; %final average of u/u_max %'s
    end
end

%% Store Paramater values
for(i=1:size(data,1))
        fx(i,1) = data(i,2);    %mu
        fx(i,2) = data(i,3);    %alpha
        fx(i,3) = data(i,4);    %eta
        fx(i,4) = data(i,5)/100; %ro (scaled)
end

%% Combine all 3 into one performance metric
% fx(mu, alpha, eta, ro scaled, convoy distance, collide, steering)

for(i=1:size(data,1))
    gx(i,1)= 2*fx(i,5)+6*fx(i,6)+2*fx(i,7);

fprintf(fidw,'%d,%s,%s,%s,%s,%s,%s,%s,%s\n',i,num2str(fx(i,1)),num2str(fx(i,2
)),num2str(fx(i,3)),num2str(fx(i,4)),num2str(fx(i,5)),num2str(fx(i,6)),num2st
r(fx(i,7)),num2str(gx(i,1)));
end
```

6.3. Minimum_Sensitivity.m

```
%%=====================================================================
% File Name    : Minimum_Sensitiviy.m
% Authors      : W. Selby
% Date         : 26-Sep-2008
% Description  : Computes cost function and varies weights
%
% Inputs:      : data text file
% Outputs:     : data text file, plot
```

```
% Requirements : ParamDistData per IP
% Revisions    : None
%=========================================================================
%% Initialize
clear; close all; format compact; clc; format short;

%% Declare variables
n = 1; %number of trials
rho = .25; % performance weighting value

%% Load Files
fidw = fopen('Min_Sens_scenario.csv','w');
fprintf(fidw,'%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s\n','Trial','Mu','Alpha','Et
a','Ro (scaled)','Convoy','Collision','E_u','Performance Average', 'Weight
1', 'Weight 2', 'Weight 3');

%% Compute 1 norm for convoy distance and steering control inf-norm for
%% collisions
j=1;
filename = 'Scenario3_output_data.csv';
    data = load(filename);
    for(i=1:size(data,1))
        if(j==1)
            fx(i+1,5:7)=0;
        end
        gx(i,1) = data(i,6);
        collide(i,j)=data(i,7);
        gx(i,2) = data(i,8);
        fx(i,5) = fx(i,5) + gx(i,1)/n; %final average of convoy distance %'s
        fx(i,6)= max(collide(i,:));
        fx(i,7) = fx(i,7) + gx(i,2)/n; %final average of u/u_max %'s
    end


%% Store Paramater values
for(i=1:size(data,1))
        fx(i,1) = data(i,2);    %mu
        fx(i,2) = data(i,3);    %alpha
        fx(i,3) = data(i,4);    %eta
        fx(i,4) = data(i,5)/100; %ro (scaled)
end

%% Combine all 3 into one performance metric
% fx(mu, alpha, eta, ro scaled, convoy distance, collide, steering)

for(w1=.1:.1:1)
    for(w2=.1:.1:(.91-w1))
        w3=1-(w1+w2);
        for(i=1:size(data,1))
            gx(i,1)= w1*fx(i,5)+w2*fx(i,6)+w3*fx(i,7);
            [min_par place] = min(gx(:,1));
        end
        H = figure(1);
        title('Sensitivity of the Test Matrix as Cost Function Weighting is
Changed (Route 1)');
```

```
        xlabel('Parameter Set'); ylabel('Performance Percentage'); xlim([1
82]);

fprintf(fidw,'%d,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s\n',place,num2str(fx(place,1
)),num2str(fx(place,2)),num2str(fx(place,3)),num2str(fx(place,4)),num2str(fx(
place,5)),num2str(fx(place,6)),num2str(fx(place,7)),num2str(gx(place,1)),num2
str(w1),num2str(w2),num2str(w3));
        sort(gx,1);
        if(w1==.1)
        plot(sort(gx(:,1),1),'blue'); hold on; grid on;
        end
        if(w1==.2)
        plot(sort(gx(:,1),1),'red'); hold on; grid on;
        end
        if(w1<=.31 && w1>=.29)
        plot(sort(gx(:,1),1),'Color',[0 .5 0]); hold on; grid on;
        end
        if(w1==.4)
        plot(sort(gx(:,1),1),'cyan'); hold on; grid on;
        end
        if(w1==.5)
        plot(sort(gx(:,1),1),'green'); hold on; grid on;
        end
        if(w1==.6)
        plot(sort(gx(:,1),1),'magenta'); hold on; grid on;
        end
        if(w1==.7)
        plot(sort(gx(:,1),1),'yellow'); hold on; grid on;
        end
        if(w1==.8)
        plot(sort(gx(:,1),1),'black'); hold on; grid on;
        end

    end

end
```

7. GPS Filtering

```
%%=====================================================================
% File Name    : GPS_filter.m
% Authors      : W. Selby
% Date         : 20-Sep-2008
% Description  : Filters GPS routes
%
% Inputs:      : data text file
% Outputs:     : filtered track file
% Requirements : track.csv
% Revisions    : None
%=====================================================================
%% Initialize
clear; close all; format compact; clc; format long;

%% Declare variables
count = 1906;
speed = 40/(60*60);  %mph->miles per second
date = '8/7/2008';
alt = 0;
delay = 0;
N = 0;
time_diff=0;


%% Load Files
fid = fopen('NRL_to_USNA_40mph.csv');
[latd, lond, time] =
textread('NRL_to_USNA_40mph.csv','%*f%f%f%*f%*s%s','delimiter',',');
fidw = fopen('to_USNA_Time_Mod.csv','w');

%% Degrees to radians conversion
for(i=1:size(latd,1))
    lat(i,1) = latd(i,1)*pi/180;
    lon(i,1) = lond(i,1)*pi/180;
end

%% Parse time string
% for(i=1:size(time,1))
%     [h(i,1), m(i,1), s(i,1)] = strread(time{i,1}, '%d%d%d',
      'delimiter', ':');
% end

h(1,1) = 8;
m(1,1) = 29;
s(1,1) = 56;

%%Print Header
% fprintf(fidw,'%s\n', 'No,Latitude,Longitude,Altitude,Date,Time');
% fprintf(fidw,'%d,%.6f,%.6f,%d,%s,%s\n',count,latd(1,1),lond(1,1),alt,
     date,time{1,1});
%% Repeat Point for loitering
if(0)
    for(i=1:300)
```

```
      %current total number of seconds and subtracts delay
      N = 86400*datenum(2008,9,21,h(1,1),m(1,1),s(1,1))+i;
      %store updated time information
      [Y, M, D, hour(i,1), min(i,1), sec(i,1)] = datevec((N/86400));
      %make new time string
      time =sprintf('%d:%d:%2.2d',hour(i,1),min(i,1),int8(sec(i,1)));

sprintf('%d,%.6f,%.6f,%d,%s,%s\n',count,latd(i,1),lond(i,1),alt,date,time);
%print line of track file
fprintf(fidw,'%d,%.6f,%.6f,%d,%s,%s\n',count,latd(i,1),lond(i,1),alt,date,tim
e);
    end
end
for(i=1:size(latd,1)-1)
    a(i,1) = sin((lat(i+1,1)-lat(i,1))/2)^2 +
cos(lat(i,1))*cos(lat(i+1,1))*sin((lon(i+1,1)-lon(i,1))/2)^2;
    c(i,1) = 2*atan2(sqrt(a(i,1)),1-sqrt(a(i,1)));
    d(i,1) = 3956*c(i,1); %convert to miles
    time_diff = (d(i,1)/speed); %time to next waypoint
    count = count+1; %line number
    %current total number of seconds and add to make future time
    N = 86400*datenum(2008,9,21,h(i,1),m(i,1),s(i,1))+time_diff;
    %store updated time information
    [Y, M, D, h(i+1,1), m(i+1,1), s(i+1,1)] = datevec((N/86400));
      if(int8(s(i+1,1))==60)
            m(i+1,1) = m(i+1,1)+1;
            s(i+1,1)=0;
        end
      %make new time string
      time = sprintf('%d:%d:%2.2d',h(i+1,1),m(i+1,1),int8(s(i+1,1)));

%print line of track file
fprintf(fidw,'%d,%.6f,%.6f,%d,%s,%s\n',count,latd(i,1),lond(i,1),alt,date,tim
e);
end

fclose('all');
```

8. Definition of Terms

- α (alpha) – parameter that controls the equidistant spacing behavior
- η (eta) – parameter that control the baseline alignment behavior
- μ (mu) – parameter that controls the heading alignment behavior
- u – output of the control law, a steering command for the vehicle
- r_0 - parameter that sets the desired vehicle spacing
- $r_{\min}$ - distance at which vehicles are assumed to have collided